DISTRIBUTED SENSOR NETWORKS

© 2005 by Chapman & Hall/CRC

CHAPMAN & HALL/CRC
COMPUTER and INFORMATION SCIENCE SERIES

Series Editor: Sartaj Sahni

PUBLISHED TITLES

HANDBOOK OF SCHEDULING: ALGORITHMS, MODELS, AND PERFORMANCE ANALYSIS
Joseph Y-T. Leung

DISTRIBUTED SENSOR NETWORKS
S. Sitharama Iyengar and Richard R. Brooks

FORTHCOMING TITLES

SPECULATIVE EXECUTION IN HIGH PERFORMANCE COMPUTER ARCHITECTURES
David Kaeli and Pen-Chung Yew

THE PRACTICAL HANDBOOK OF INTERNET COMPUTING
Munindar P. Singh

HANDBOOK OF DATA STRUCTURES AND APPLICATIONS
Dinesh P. Mehta and Sartaj Sahni

© 2005 by Chapman & Hall/CRC

CHAPMAN & HALL/CRC COMPUTER and INFORMATION SCIENCE SERIES

DISTRIBUTED SENSOR NETWORKS

Edited by

S. Sitharama Iyengar

ACM Fellow, IEEE Fellow, AAAS Fellow

Roy Paul Daniels Professor of Computer Science and Chairman

Department of Computer Science

Louisiana State University

and

Richard R. Brooks

Associate Professor

Holcombe Department of Electrical and

Computer Engineering

Clemson University



CHAPMAN & HALL/CRC

A CRC Press Company

Boca Raton London New York Washington, D.C.

© 2005 by Chapman & Hall/CRC

Library of Congress Cataloging-in-Publication Data

Catalog record is available from the Library of Congress

This book contains information obtained from authentic and highly regarded sources. Reprinted material is quoted with permission, and sources are indicated. A wide variety of references are listed. Reasonable efforts have been made to publish reliable data and information, but the author and the publisher cannot assume responsibility for the validity of all materials or for the consequences of their use.

Neither this book nor any part may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, microfilming, and recording, or by any information storage or retrieval system, without prior permission in writing from the publisher.

All rights reserved. Authorization to photocopy items for internal or personal use, or the personal or internal use of specific clients, may be granted by CRC Press, provided that $1.50 per page photocopied is paid directly to Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923 USA. The fee code for users of the Transactional Reporting Service is ISBN 1-58488-383-9/05/$0.00 + $1.50. The fee is subject to change without notice. For organizations that have granted a photocopy license by the CCC, a separate system of payment has been arranged.

The consent of CRC Press does not extend to copying for general distribution, for promotion, for creating new works, or for resale. Specific permission must be obtained in writing from CRC Press for such copying.

Direct all inquiries to CRC Press, 2000 N.W. Corporate Blvd., Boca Raton, Florida 33431.

Trademark Notice: Product or corporate names may be trademarks or registered trademarks, and are used only for identification and explanation, without intent to infringe.

Visit the CRC Press Web site at www.crcpress.com

© 2005 by Chapman & Hall/CRC

No claim to original U.S. Government works
International Standard Book Number 1-58488-383-9
Printed in the United States of America 1 2 3 4 5 6 7 8 9 0
Printed on acid-free paper

© 2005 by Chapman & Hall/CRC

Dedicated to Dr. S.S. Iyengar and Dr. S. Rai of LSU, whose ongoing mentoring has always been appreciated.

— R.R. Brooks

Dedicated to all my former/present Graduate and Undergraduate Students; to Prof. Kasturirangan, former ISRO Chairman, towards his dedication to Space Technology; Prof. Hartamanis and Prof. C.N.R. Rao for their inspiring research, and to Vice Provost Harold Silverman for providing an environment and mentoring me at different stages of my career.

— S.S. Iyengar

© 2005 by Chapman & Hall/CRC

Preface

In many ways this book started 10 years ago, when the editors started their collaboration at Louisiana State University in Baton Rouge. At that time, sensor networks were a somewhat arcane topic. Since then, many new technologies have ripened, and prototype devices have emerged on the market. We were lucky enough to be able to continue our collaboration under the aegis of the DARPA IXO Sensor Information Technology Program, and the Emergent Surveillance Plexus Multidisciplinary University Research Initiative.

What was clear 10 years ago, and has become more obvious since, is that the only way to monitor the real world adequately is to use a network of devices. Many reasons for this will be given in this book. These reasons range from financial considerations to statistical inference constraints. Once you start using a network situated in the real world, the need for adaptation and self-configuration also become obvious.

What was probably not known 10 years ago was the breadth and depth of research needed to design these systems adequately. The book in front of you contains chapters from acknowledged leaders in sensor network design. The contributors work at leading research institutions and have expertise in a broad range of technical fields.

The field of sensor networks has matured greatly within the last few years. The editors are grateful to have participated in this process. We are especially pleased to have been able to interact with the research groups whose work is presented here. This growth has only been possible with the support from many government agencies, especially within the Department of Defense. Visionary program managers at DARPA, ONR, AFRL, and ARL have made a significant impact on these technologies.

It is the editors' sincere hope that the field continues to mature. We also hope that the cross-fertilization of ideas between technical fields that has enabled these advances, deepens.

© 2005 by Chapman & Hall/CRC

Contributors

Mohiuddin Ahmed
Electrical Engineering Department
University of California
Los Angeles, California

N. Balakrishnan
Supercomputing Research Center
Indian Institute of Science
Bangalore, India

Steve Beck
BAE Systems, IDS
Austin, Texas

Edo Biagioni
Department of Information and Computer Sciences
University of Hawaii at Manoa
Honolulu, Hawaii

N. K. Bose
Department of Electrical Engineering
The Pennsylvania State University
University Park, Pennsylvania

Cliff Bowman
Ember Corporation
Boston, Massachusetts

K. W. Bridges
Department of Botany
University of Hawaii at Manoa
Honolulu, Hawaii

R. R. Brooks
Holcombe Department of Electrical and Computer Engineering
Clemson University
Clemson, South Carolina

David W. Carman
McAfee Research
Rockville, Maryland

Krishnendu Chakrabarty
Department of Electrical and Computer Engineering
Duke University
Durham, North Carolina

G. Chen
Microsystems Design Laboratory
The Pennsylvania State University
University Park, Pennsylvania

J. C. Chen
Electrical Engineering Department
University of California
Los Angeles, California

Eungchun Cho
Division of Mathematics and Sciences
Kentucky State University
Frankfort, Kentucky

A. Choudhary
Department of ECE
Northwestern University
Evanston, Illinois

Eiman Elnahrawy
Department of Computer Science
Rutgers University
Rutgers, New Jersey

Deborah Estrin
Information Sciences Institute
University of Southern California
Marina del Rey, California
and
Computer Science Department
University of California
Los Angeles, California

D. S. Friedlander
Applied Research Laboratory
The Pennsylvania State University
State College, Pennsylvania

N. Gautam
The Pennsylvania State University
University Park, Pennsylvania

Johannes Gehrke
University of California
Berkeley, California
and
Cornell University
Ithaca, New York

© 2005 by Chapman & Hall/CRC

Ramesh Govindan
Information Sciences Institute
University of Southern California
Marina del Rey, California
and
Computer Science Department
University of Southern California
Los Angeles, California

Lynne Grewe
Department of Mathematics and Computer Science
California State University
Hayward, California

C. Griffin
Applied Research Laboratory
The Pennsylvania State University
State College, Pennsylvania

Leonidas Guibas
Computer Science Department
Stanford University
Stanford, California

David L. Hall
The Pennsylvania State University
University Park, Pennsylvania

John Heidemann
Information Sciences Institute
University of Southern California
Marina del Rey, California

Yu Hen Hu
Department of Electrical and Computer Engineering
University of Wisconsin
Madison, Wisconsin

M. J. Irwin
Microsystems Design Laboratory
The Pennsylvania State University
University Park, Pennsylvania
and
Computer Science and Engineering
Applied Research Laboratory
The Pennsylvania State University
State College, Pennsylvania

S. S. Iyengar
Department of Computer Science
Louisiana State University
Baton Rouge, Louisiana

Vijay S. Iyer
Supercomputing Research Center
Indian Institute of Science
Bangalore, India

I. Kadayif
Microsystems Design Laboratory
The Pennsylvania State University
University Park, Pennsylvania

M. Kandemir
Microsystems Design Laboratory
The Pennsylvania State University
University Park, Pennsylvania
and
Computer Science and Engineering
Applied Research Laboratory
The Pennsylvania State University
State College, Pennsylvania

B. Kang
Microsystems Design Laboratory
The Pennsylvania State University
University Park, Pennsylvania

Rajgopal Kannan
Department of Computer Science
Louisiana State University
Baton Rouge, Louisiana

M. Karakoy
Department of Computing
Imperial College
University of London
London, UK

T. Keiser
Distributed Systems Department
Applied Research Laboratory
The Pennsylvania State University
State College, Pennsylvania

J. D. Koch
Applied Research Laboratory
The Pennsylvania State University
State College, Pennsylvania

Richard J. Kozick
Department of Electrical Engineering
Bucknell University
Lewisburg, Pennsylvania

Bhaskar Krishnamachari
Department of Electrical Engineering
University of Southern California
Los Angeles, California

Teja Phani Kuruganti
Electrical and Computer Engineering Department
University of Tennessee
Knoxville, Tennessee

Jacob Lamb
Distributed Systems Department
Applied Research Laboratory
The Pennsylvania State University
State College, Pennsylvania

L. Li
Microsystems Design Laboratory
The Pennsylvania State University
University Park, Pennsylvania

Alvin S. Lim
Department of Computer Science and Engineering
Auburn University
Auburn, Alabama

Jie Liu
Palo Alto Research Center (PARC)
Palo Alto, California

Juan Liu
Palo Alto Research Center (PARC)
Palo Alto, California

Samuel Madden
University of California
Berkeley, California
and
Cornell University
Ithaca, New York

Prakash Manghwani
BBN Technologies
Cambridge, Massachusetts

© 2005 by Chapman & Hall/CRC

Jeff Mazurek
BBN Technologies
Cambridge, Massachusetts

Gail Mitchell
BBN Technologies
Cambridge, Massachusetts

Badri Nath
Department of Computer Science
Rutgers University
Rutgers, New Jersey

Shashi Phoha
Applied Research Laboratory
The Pennsylvania State University
State College, Pennsylvania

Matthew Pirretti
Distributed Systems Department
Applied Research Laboratory
The Pennsylvania State University
State College, Pennsylvania

Robert Poor
Ember Corporation
Boston, Massachusetts

Gregory Pottie
Electrical Engineering Department
University of California
Los Angeles, California

Hairong Qi
Department of Electrical and Computer Engineering
The University of Tennessee
Knoxville, Tennessee

Suresh Rai
Department of Electrical and Computer Engineering
Louisiana State University
Baton Rouge, Louisiana

Parameswaran Ramanathan
Department of Electrical and Computer Engineering
University of Wisconsin
Madison, Wisconsin

Nageswara S. V. Rao
Computer Science and Mathematics Division
Center for Engineering Science Advance Research
Oak Ridge National Laboratory
Oak Ridge, Tennessee

Asok Ray
Mechanical Engineering Department
The Pennsylvania State University
University Park, Pennsylvania

James Reich
Palo Alto Research Center (PARC)
Palo Alto, California

Brian M. Sadler
Army Research Laboratory
Adelphi, Maryland

Prince Samar
School of Electrical and Computer Engineering
Cornell University
Ithaca, New York

H. Saputra
Computer Science and Engineering
Applied Research Laboratory
The Pennsylvania State University
University Park, Pennsylvania

Shivakumar Sastry
Department of Electrical and Computer Engineering
The University of Akron
Akron, Ohio

Akbar M. Sayeed
Department of Electrical and Computer Engineering
University of Wisconsin
Madison, Wisconsin

Ben Shahshahani
Nuance Communications
Menlo Park, California

David Shepherd
SA Inc.

Fabio Silva
Information Sciences Institute
University of Southern California
Marina del Rey, California

Vishnu Swaminathan
Department of Electrical and Computer Engineering
Duke University
Durham, North Carolina

David C. Swanson
Applied Research Laboratory
The Pennsylvania State University
State College, Pennsylvania

Ankit Tandon
Department of Computer Science
Louisiana State University
Baton Rouge, Louisiana

Ken Theriault
BBN Technologies
Cambridge, Massachusetts

Vijay K. Vaishnavi
Department of Computer Information Systems
Georgia State University
Atlanta, Georgia

N. Vijaykrishnan
Microsystems Design Laboratory
The Pennsylvania State University
University Park, Pennsylvania
and
Computer Science and Engineering
Applied Research Laboratory
The Pennsylvania State University
State College, Pennsylvania

Kuang-Ching Wang
Department of Electrical and Computer Engineering
University of Wisconsin
Madison, Wisconsin

Xiaoling Wang
Department of Electrical and Computer Engineering
The University of Tennessee
Knoxville, Tennessee

© 2005 by Chapman & Hall/CRC

Stephen B. Wicker
School of Electrical and Computer Engineering
Cornell University
Ithaca, New York

D. Keith Wilson
U.S. Army Cold Regions Research and Engineering Laboratory
Hanover, New Hampshire

Qishi Wu
Computer Science and Mathematics Division
Oak Ridge National Laboratory
Oak Ridge, Tennessee
and
Department of Computer Science
Louisiana State University
Baton Rouge, Louisiana

Yingyue Xu
Electrical and Computer Engineering Department
University of Tennessee
Knoxville, Tennessee

K. Yao
Electrical Engineering Department
University of California
Los Angeles, California

Feng Zhao
Palo Alto Research Center (PARC)
Palo Alto, California

Mengxia Zhu
Department of Computer Science
Louisiana State University
Baton Rouge, Louisiana

Yi Zou
Department of Electrical and Computer Engineering
Duke University
Durham, North Carolina

© 2005 by Chapman & Hall/CRC

Contents

SECTION I: OVERVIEW 1

Chapter 1 An Overview 3
S.S. Iyengar, Ankit Tandon, and R.R. Brooks

Chapter 2 Microsensor Applications 11
David Shepherd and Sri Kumar

Chapter 3 A Taxonomy of Distributed Sensor Networks 29
Shivakumar Sastry and S.S. Iyengar

Chapter 4 Contrast with Traditional Systems 45
R.R. Brooks

SECTION II: DISTRIBUTED SENSING AND SIGNAL PROCESSING 49

Chapter 5 Digital Signal Processing Backgrounds 53
Yu Hen Hu

Chapter 6 Image-Processing Background 71
Lynne Grewe and Ben Shahshahani

Chapter 7 Object Detection and Classification 97
Akbar M. Sayeed

Chapter 8 Parameter Estimation 115
David S. Friedlander

Chapter 9 Target Tracking with Self-Organizing Distributed Sensors 135
R.R. Brooks, C. Griffin, David S. Friedlander, and J.D. Koch

Chapter 10 Collaborative Signal and Information Processing: An Information-Directed Approach 185
Feng Zhao, Jie Liu, Juan Liu, Leonidas Guibas, and James Reich

Chapter 11 Environmental Effects 201
David C. Swanson

Chapter 12 Detecting and Counteracting Atmospheric Effects 213
Lynne L. Grewe

© 2005 by Chapman & Hall/CRC

Chapter 13 **Signal Processing and Propagation for Aeroacoustic Sensor Networks** 225
Richard J. Kozick, Brian M. Sadler, and D. Keith Wilson

Chapter 14 **Distributed Multi-Target Detection in Sensor Networks** 271
Xiaoling Wang, Hairong Qi, and Steve Beck

SECTION III: INFORMATION FUSION 287

Chapter 15 **Foundations of Data Fusion for Automation** 291
S.S. Iyengar, S. Sastry, and N. Balakrishnan

Chapter 16 **Measurement-Based Statistical Fusion Methods For Distributed Sensor Networks** 301
Nageswara S.V. Rao

Chapter 17 **Soft Computing Techniques** 321
R.R. Brooks

Chapter 18 **Estimation and Kalman Filters** 335
David L. Hall

Chapter 19 **Data Registration** 361
R.R. Brooks, Jacob Lamb, and Lynne Grewe

Chapter 20 **Signal Calibration, Estimation for Real-Time Monitoring and Control** 391
Asok Ray and Shashi Phoha

Chapter 21 **Semantic Information Extraction** 409
David S. Friedlander

Chapter 22 **Fusion in the Context of Information Theory** 419
Mohiuddin Ahmed and Gregory Pottie

Chapter 23 **Multispectral Sensing** 437
N.K. Bose

SECTION IV: SENSOR DEPLOYMENT AND NETWORKING 449

Chapter 24 **Coverage-Oriented Sensor Deployment** 453
Yi Zou and Krishnendu Chakrabarty

Chapter 25 **Deployment of Sensors: An Overview** 483
S.S. Iyengar, Ankit Tandon, Qishi Wu, Eungchun Cho, Nageswara S.V. Rao, and Vijay K. Vaishnavi

Chapter 26 **Genetic Algorithm for Mobile Agent Routing in Distributed Sensor Networks** 505
Qishi Wu, S.S. Iyengar, and Nageswara S.V. Rao

Chapter 27 **Computer Network — Basic Principles** 527
Suresh Rai

Chapter 28 **Location-Centric Networking in Distributed Sensor Networks** 555
Kuang-Ching Wang and Parameswaran Ramanathan

Chapter 29 **Directed Diffusion** 573
Fabio Silva, John Heidemann, Ramesh Govindan, and Deborah Estrin

© 2005 by Chapman & Hall/CRC

Chapter 30 Data Security Perspectives **597**
David W. Carman

Chapter 31 Quality of Service Metrics **613**
N. Gautam

Chapter 32 Network Daemons for Distributed Sensor Networks **629**
Nageswara S.V. Rao and Qishi Wu

SECTION V: POWER MANAGEMENT **651**

Chapter 33 Designing Energy-Aware Sensor Systems **653**
N. Vijaykrishnan, M.J. Irwin, M. Kandemir, L. Li, G. Chen, and B. Kang

Chapter 34 Operating System Power Management **667**
Vishnu Swaminathan and Krishnendu Chakrabarty

Chapter 35 An Energy-Aware Approach for Sensor Data Communication **697**
H. Saputra, N. Vijaykrishnan, M. Kandemir, R.R. Brooks, and M.J. Irwin

Chapter 36 Compiler-Directed Communication Energy Optimizations for Microsensor Networks **711**
I. Kadayif, M. Kandemir, A. Choudhary, M. Karakoy, N. Vijaykrishnan, and M.J. Irwin

Chapter 37 Sensor-Centric Routing in Wireless Sensor Networks **735**
Rajgopal Kannan and S.S. Iyengar

SECTION VI: ADAPTIVE TASKING **749**

Chapter 38 Query Processing in Sensor Networks **751**
Samuel Madden and Johannes Gehrke

Chapter 39 Autonomous Software Reconfiguration **773**
R.R. Brooks

Chapter 40 Mobile Code Support **787**
R.R. Brooks and T. Keiser

Chapter 41 The Mobile-Agent Framework for Collaborative Processing in Sensor Networks **801**
Hairong Qi, Yingyue Xu, and Teja Phani Kuruganti

Chapter 42 Distributed Services **819**
Alvin S. Lim

Chapter 43 Adaptive Active Querying **835**
Bhaskar Krishnamachari

SECTION VII: SELF-CONFIGURATION **845**

Chapter 44 Need for Self-Configuration **847**
R.R. Brooks

Chapter 45 Emergence **855**
R.R. Brooks

Chapter 46 Biological Primitives **863**
M. Pirretti, R.R. Brooks, J. Lamb, and M. Zhu

© 2005 by Chapman & Hall/CRC

Chapter 47 Physics and Chemistry 879
Mengxia Zhu, Richard Brooks, Matthew Pirretti, and S.S. Iyengar

Chapter 48 Collective Intelligence for Power-Aware Routing in Mobile *Ad Hoc* Sensor Networks 895
Vijay S. Iyer, S.S. Iyengar, and N. Balakrishnan

Chapter 49 Random Networks and Percolation Theory 907
R.R. Brooks

Chapter 50 On the Behavior of Communication Links in a Multi-Hop Mobile Environment 947
Prince Samar and Stephen B. Wicker

SECTION VIII: SYSTEM CONTROL 975

Chapter 51 Example Distributed Sensor Network Control Hierarchy 977
Mengxia Zhu, S.S. Iyengar, Jacob Lamb, R.R. Brooks, and Matthew Pirretti

SECTION IX: ENGINEERING EXAMPLES 1009

Chapter 52 SenSoft: Development of a Collaborative Sensor Network 1011
Gail Mitchell, Jeff Mazurek, Ken Theriault, and Prakash Manghwani

Chapter 53 Statistical Approaches to Cleaning Sensor Data 1023
Eiman Elnahrawy and Badri Nath

Chapter 54 Plant Monitoring with Special Reference to Endangered Species 1039
K.W. Bridges and Edo Biagioni

Chapter 55 Designing Distributed Sensor Applications for Wireless Mesh Networks 1049
Robert Poor and Cliff Bowman

SECTION X: BEAMFORMING 1067

Chapter 56 Beamforming 1069
J.C. Chen and K. Yao

© 2005 by Chapman & Hall/CRC

I

Overview

1. **An Overview** *S.S. Iyengar, Ankit Tandon and R.R. Brooks* 3
   Introduction • Example Applications • Computing Issues in Sensor Networks • Requirements of Distributed Sensor Networks • Communications in Distributed Sensor Networks • Mobile-Agent Paradigm • Technology Needed • Contrast with Traditional Computing Systems
2. **Microsensor Applications** *David Shepherd* 11
   Introduction • Sensor Networks: Description • Sensor Network Applications, Part 1: Military Applications • Sensor Network Applications, Part 2: Civilian Applications • Conclusion
3. **A Taxonomy of Distributed Sensor Networks** *Shivakumar Sastry and S.S. Iyengar* 29
   Introduction • Benefits and Limitations of DSNs • General Technology Trends Affecting DSNs • Taxonomy of DSN Architectures • Conclusions • Acknowledgments
4. **Contrast with Traditional Systems** *R.R. Brooks* 45
   Problem Statement • Acknowledgments and Disclaimer

This section provides a brief overview of sensor networks. It introduces the topics by discussing what they are, their applications, and how they are differ from traditional systems.

Iyengar *et al.* provide a definition of distributed sensor networks (DSNs). They introduce many applications that will be dealt with in more detail later. A discussion is also provided of the technical challenges these systems present.

Kumar provides an overview of sensor networks from the military perspective. Of particular interest is a summary of military applications starting in the 1960s. This chapter then proceeds to recent research advances. Many of these advances come from research groups presented in later sections of this book.

Sastry and Iyengar provide a taxonomy of DSNs. The taxonomy should help readers in structuring their view of the field. It also is built on laws describing the evolution of technology. These laws can help readers anticipate the future developments that are likely to appear in this domain.

© 2005 by Chapman & Hall/CRC

Brooks describes briefly how DSNs differ from traditional systems. The global system is composed of distributed elements that are failure prone and have a limited lifetime. Creating a reliable system from these components requires a new type of flexible system design.

The purpose of this section is to provide a brief overview of DSNs. The chapters presented concentrate on the applications of this technology and why the new technologies presented in this book are necessary.

© 2005 by Chapman & Hall/CRC

1 An Overview

S.S. Iyengar, Ankit Tandon, and R.R. Brooks

1.1 Introduction

In a recent statement, Mr. Donald Rumsfeld, the U.S. Secretary of Defense, said:

> A revolution in military affairs is about more than building new high-tech weapons, though that is certainly part of it. It's also about new ways of thinking . . .
>
> New concepts and techniques are taking form in our defense analysis methodology. These new concepts have their motivation in the defense debate of today. That debate ponders issues involving the reaction of adaptive threats, the consequences of effects based operations, the modes and value of information operations, the structure and performance of command and control, and a host of other difficult to analyze subjects.

Since the early 1990s, distributed sensor networks (DSNs) have been an area of active research. The trend is to move from a centralized, super-reliable single-node platform to a dense and distributed multitude of cheap, lightweight and potentially individually unreliable components that, as a group, are capable of far more complex tasks and inferences than any individual super-node [1]. An example of such system is a DSN. Such distributed systems are displacing the more traditional centralized architectures at a prodigious rate.

A DSN is a collection of a large number of heterogeneous intelligent sensors distributed logically, spatially, or geographically over an environment and connected through a high-speed network. The sensors may be cameras as vision sensors, microphones as audio sensors, ultrasonic sensors, infrared sensors, humidity sensors, light sensors, temperature sensors, pressure/force sensors, vibration sensors, radioactivity sensors, seismic sensors, etc. Figure 1.1 shows a diagram of a DSN.

The sensors continuously collect measurement data from their respective environments. The data collected are processed by an associated processing element that then transmits it through an interconnected communication network. The information that is gathered from all other parts of the sensor network is then integrated using some data-fusion strategy. This integrated information is then useful to derive appropriate inferences about the environment in which the sensors are deployed. Figure 1.2 shows a networking structure of a DSN.

These sensors may be distributed in a two-dimensional (2-D) or a three-dimensional (3-D) environment. The environment in which these sensors are deployed varies with the application.

© 2005 by Chapman & Hall/CRC

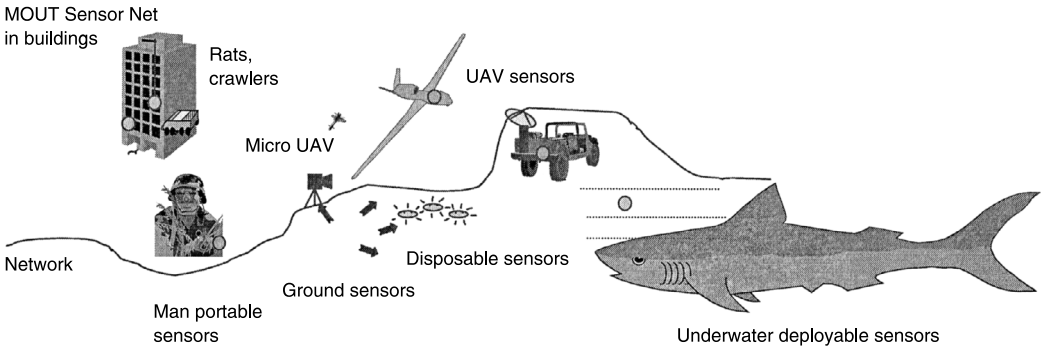


Figure 1.1. A typical DSN.

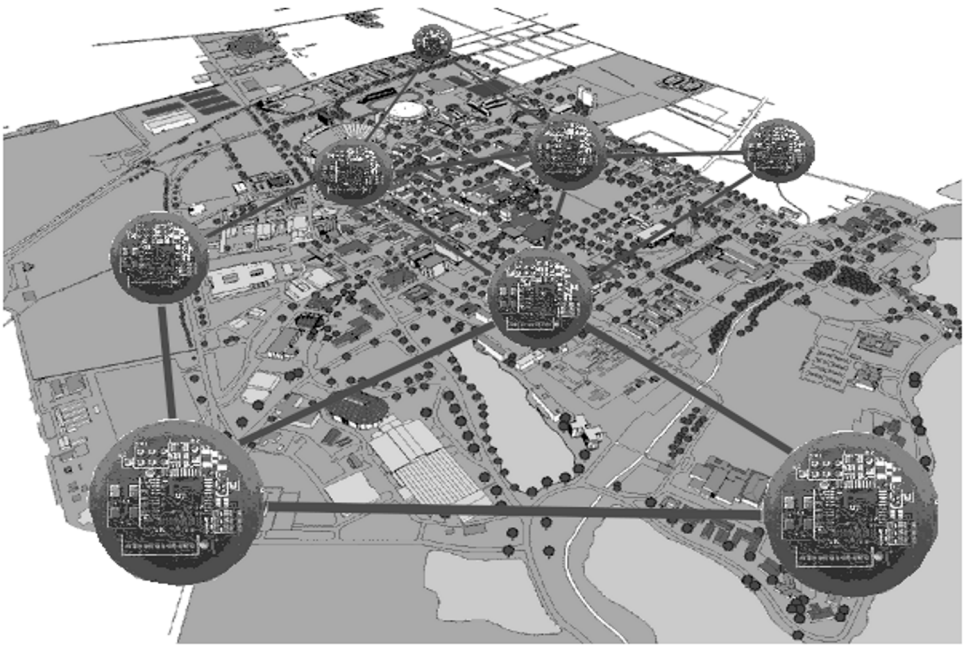


Figure 1.2. Networking structure of a DSN.

For example, it may be enemy terrain for reconnaissance or information gathering, in a forest for ecological monitoring, or inside a nuclear power plant for detecting radiation levels.

1.2 Example Applications

With the emergence of high-speed networks and with their increased computational capabilities, DSNs have a wide range of real-time applications in aerospace, automation, defense, medical imaging, robotics, weather prediction, etc. To elucidate, let us consider sensors spread in a large geographical territory collecting data on various parameters like temperature, atmospheric pressure, wind velocity, etc. The data from these sensors are not as useful when studied individually, but when integrated, the data give the picture of a large area. Changes in the data across time for the entire region can be used in predicting the weather at a particular location.

Modern battle spaces have become technologically very large and complex. Information must be collected and put into comprehensible form. Algorithms are needed to study postulated battle-space

© 2005 by Chapman & Hall/CRC

environments to reduce them into fundamental information components. Then, algorithms are needed to provide real-time elemental information in a concise format in actual deployment. Algorithms must adapt to new patterns in the data and provide feedback to the collection process. Military applications require algorithms with great correctness and precision, and must work with limited or incomplete information.

Another scenario where DSNs may be useful is for intrusion detection: a number of different types of sensor may be placed at the perimeter of a secure location, like a manufacturing plant or some other guarded location of similar sort. Yet another example is in multimedia and hearing aids. Sensors that are capable of detecting noises may be placed at various locations inside an auditorium. The sensor network would then enhance audio signals, ensuring improved intelligibility under noisy conditions.

Consider a (3-D) scene reconstruction system where a number of cameras, placed in different locations in a room, act as vision sensors. The 2-D images from these cameras are transmitted to a central base-system that uses an object visualization algorithm to create a 3-D approximation of the scene. The system functions like the compound vision system found in some species of insect.

Such 3-D scene reconstruction systems may be extended to other applications as well. Satellites may be used to perform remote sensing and the data gathered from them can be used to construct 3-D topological maps of territories that are otherwise inaccessible, like enemy terrain or even deep space. Any and all information amassed beforehand is useful before exploring such unknowns.

In hospitals, doctors may use tiny biological sensors that are either placed in the blood stream of a person or at various exterior locations. These sensors can continuously measure blood pressure, pulse rate, temperature, sugar level, hormone levels, etc. and send the data to a patient-monitoring system automatically where it can be integrated and make inferences in real time. The system may notify the nearest available doctor when an abrupt or undesirable change occurs in the data being collected.

Another example is the humidity and temperature sensors in a building. These collect data from various areas in the building and transmit it to a central system. This system then uses the data to regulate the air-conditioners and humidifiers in the building, maintaining the desired ambience.

An object-positioning system can be implemented in large offices and supermarkets. Every object that needs to be located is tagged with an active badge that emits unique infrared codes. Sensors dispersed at various locations inside the building pick up these codes. Depending on the time of flight of a round-trip signal, the distance of the object from each receiving sensor is calculated. The position of the object can then be computed using superposition techniques.

Sensors that coordinate with a global positioning system may also be placed on vehicles to pinpoint their location at any time. The sensors, when coupled to a traffic monitoring system, would provide data to enable more effective regulation of traffic through congested areas of a city.

A manufacturing plant can place small cameras along its automated assembly line to inspect the product. For example, using a number of tiny cameras, a car manufacturing plant could detect whether the paint job on all its cars is uniform. The sensors would transmit their data to a central location that would accept or reject a painted car using the data.

Pressure sensors embedded at various points in the structure of a building can measure stress levels. This information can be of substantial help to civil engineers in fixing unforeseen design errors and would prevent avoidable casualties. A similar application relates to a seismic activity detection system, where a number of seismic sensors are placed at various locations in the ground. Raw vibration data are sent to a central location where it can be studied to distinguish footsteps and heavy vehicles from earthquakes. The data can also be used to calculate and record the intensities and epicenters of earthquakes for a region over time.

1.3 Computing Issues in Sensor Networks

Distributed, real-time sensor networks are essential for effective surveillance in the digitized battlefield and environmental monitoring. An important issue in the design of these networks is the underlying

© 2005 by Chapman & Hall/CRC

theoretical framework and the corresponding efficient algorithms for optimal information analysis in the sensor field. The key challenge here is to develop network models and computationally efficient approaches for analyzing sensor information.

A primary concern is the layout or distribution of sensors in the environment. The number, type, location and density of sensors determine the layout of a sensor network. An intelligent placement of sensors can enhance the performance of the system significantly. Some redundancy is needed for error detection and for the correction of faulty sensors and an unreliable communication network. However, large numbers of sensors correspond to higher deployment costs, the need for higher bandwidth, increased collisions in relaying messages, higher energy consumption and more time-consuming algorithms for data fusion. Thus, sensor placement with appropriate grid coverage and optimum redundancy needs further study.

Since the sensors may be randomly distributed in widespread hazardous, unreliable, or possibly even adversarial environments, it is essential that they do not require human attention very often. Usually, the sensors are self-aware, self-reconfigurable, and autonomous, collecting data and transmitting it by themselves. Unlike laptops or other handheld devices that require constant attention and maintenance by humans, the scale of a sensor network deployment makes replenishment of energy reserves impossible. Hence, sensors have to be self-powered with rechargeable batteries. Power in each sensor is thus finite and precious, and it is extremely essential to conserve it.

Sensor networks are extremely useful because of their ability to function by themselves. It is essential to make the sensor nodes intelligent enough to adapt to changes in the network topology, node failures in the surroundings, and network degradation. Since the sensors are typically deployed in harsh environments, these abilities can be of high importance.

The advancement of micro-fabrication has revolutionized the integration of mechanical elements, sensors, actuators, and electronics on a common silicon substrate. Using this technology, a number of micro-electro-mechanical systems with unprecedented levels of functionality may now be mounted on small chips at relatively low cost. The advances in technology have made it possible for these small, but smart sensors to carry out minor processing of data before transmitting it. Furthermore, with low-power digital and analog electronic devices, and low-power radio-frequency designs, the development of building relatively inexpensive smart micro-sensors has become possible. These technological advances have not only opened up many possibilities, but also have introduced a plethora of challenging issues that call for collaborative research.

Sensors in a DSN typically communicate through wireless networks where the bandwidth is significantly lower than the wired channels. Wireless networks are more unreliable and data-faulty; therefore, there is a need for robust, fault-tolerant routing and data-fusion algorithms. It is of utmost importance to use techniques that increase the efficiency of data communication, reducing the number of overall bits transmitted and also reducing the number of unnecessary collisions. In turn, this would make the system more energy efficient.

The efficiency of algorithms that collect and analyze the sensor information also determines how a system functions. These algorithms define the path of information flow and the type of information. Simulations have shown that it typically requires around 100 to 1000 times more energy to transmit a bit than to execute an instruction [2], which means that it is beneficial to compress the data before transmitting it. These algorithms determine the processing of the sensor and the amount of information it has to transmit. It is essential to minimize data transfer in the sensor network so that the system is energy efficient.

The multi-sensor data-fusion algorithms also determine whether the data transmission is going to be proactive or reactive. Proactive means table-driven; that is, each sensor is fully aware of its surroundings and maintains up-to-date routing tables with paths from every node to every other node in the network. The storage requirement of the routing tables and the transmission overhead of any topology change are the main drawbacks of this protocol. In contrast, reactive algorithms work on the demand of a source node to transmit data to a destination node.

© 2005 by Chapman & Hall/CRC

Another approach in collecting and integrating data from sensors is mobile-agent based, and a DSN employing such a scheme is termed a MADSN. Generally speaking, a mobile agent is a special kind of software that can execute autonomously. Once dispatched, it migrates from node to node and performs processing for the node and collects data. Mobile agents reduce network load, overcome network latency, and provide robust and fault-tolerant performance. However, they raise security concerns. As mobile agents have to travel from node to node, a security system needs to be implemented whereby the agent identifies itself before it is given access to the data by the sensor node.

The routing and data-fusion algorithms also have to cope with the randomness of the sensor deployment. The sensors in a real environment start up dynamically. Their random failures also need to be taken care of by the routing and data-fusion algorithms. Consider a group of sensors that are parachuted into a forest in enemy terrain. The sensors may fail due to natural causes, component wear-out, power failures, and radio jamming in a war-case scenario. In such cases of node failures, there may be some disconnected sensors and sensor pairs that need a larger number of hops to communicate with each other. In such a military application, there is sometimes a requirement to work with incomplete data. Hence, fault-tolerant and real-time adaptive routing schemes need to be researched and implemented for such strategic applications.

For real-time medical and military applications, it is sometimes essential to have an estimate of the message delay between two nodes of a sensor network. Researchers have devised algorithms to compute the message delay given the probability of node failures, an approximate diameter of the network and its layout. However, these algorithms are computationally very expensive and provide a challenge for further study.

1.4 Requirements of Distributed Sensor Networks

A DSN is basically a system of connected, cooperating, generally diverse sensors that are spatially dispersed. The major task of a DSN is to process data, possibly noise-corrupted, acquired by the various sensors and to integrate it, reduce the uncertainty in it, and produce abstract interpretations of it. Three important facts emerge from such a framework:

1. The network must have intelligence at each node.
2. It must accommodate diverse sensors.
3. Its performance must not degrade because of spatial distribution.

DSNs are assumed to function under the following conditions:

1. Each sensor in the ensemble can see some, but not all, of the low-level activities performed by the sensor network as a whole.
2. Data are perishable, in the sense that information value depends critically upon the time required to acquire and process it.
3. There should be limited communication among the sensor processors, so that a communication–computation trade-off can be made.
4. There should be sufficient information in the system to overcome certain adverse conditions (e.g. node and link failures) and still arrive at a solution in its specific problem domain.

The successful integration of multiple, diverse sensors into a useful sensor network requires the following:

1. The development of methods to abstractly represent information gained from sensors so that this information may easily be integrated.
2. The development of methods to deal with possible differences in points of view on frames of reference between multiple sensors.

© 2005 by Chapman & Hall/CRC

3. The development of methods to model sensor signals so that the degree of uncertainty is reduced.

1.5 Communications in Distributed Sensor Networks

In a typical DSN, each node needs to fuse the local information with the data collected by the other nodes so that an updated assessment is obtained. Current research involves fusion based on a multiple hypothesis approach. Maintaining consistency and eliminating redundancy are the two important considerations. The problem of determining what should be communicated is more important than how this communication is to be effected. An analysis of this problem yields the following classes of information as likely candidates for being communicated: information about the DSN; information about the state of the world, hypothesis, conjectures and special requests for specifics actions. It is easy to see that different classes of information warrant different degrees of reliability and urgency. For further details regarding this information, see References [3–8].

1.6 Mobile-Agent Paradigm

DSNs can be of two types, consisting of either mobile sensors or immobile sensors. Normally, immobile sensors are used because they form a network that is less intricate. A unit consisting of a processing element and all its associated sensors is termed a node. The data sensed by sensor nodes in a DSN may not be of much significance individually. For example, consider the seismic detection system mentioned above. The raw data from a sensor may trigger false alarms because the sensor does not distinguish vibrations caused by a heavy vehicle from vibrations caused by an actual earthquake. In such a case, it is desired to integrate data received from sensors deployed over a larger region and then derive appropriate inferences.

Each node in the sensor network contains a number of sensors, a small amount of memory, signal processing engines, and a wireless communications link, powered by a battery. The nodes transmit the data they collect to a central location, where it is integrated, stored, and used. Data packets are sent either to the data sink directly or through a sequence of intermediate nodes. Because of the limitation of radio transmission range and energy efficiency considerations, sensors typically coordinate with the nearby nodes to forward their data.

The network topology determines what routing scheme is used. In general, two major routing approaches have been considered for sensor networks: flat multi-hop and clustering. Generally, in the flat multi-hop scheme, data are sent through the shortest path between the source and the destination. Intermediary nodes act as repeaters and simply relay messages. It is difficult to adapt to topology changes in such a scheme.

In contrast, a clustering algorithm is re-executed in the event of topology changes. Clusters are formed of a group of closely located nodes and each cluster has a head. Cluster heads are responsible for inter-cluster transmissions and they usually have a longer range radio and sophisticated location awareness. The nodes that do not belong to any cluster are called gateways, and these form a virtual backbone to relay the inter-cluster messages through them.

A DSN usually uses a clustering protocol because it has a topology that is analogous to examples in the real world. For example, consider sensors that detect levels of air pollution distributed all throughout a state. As there is more activity and more vehicles in densely populated areas, the distribution of sensors would not be uniform. Sensors in the cities would form clusters with higher density, whereas sensors in the farmlands between cities can form the backbone with gateway nodes.

A 2-D or 3-D grid of points (coordinates) is used to model the field of a sensor network. Each point represents a sensor node and each edge represents that the two sensors are in range and that direct message transmission between them is possible. This representation is like a directed graph, where each vertex is a sensor and each edge represents that the two sensors are in range; that is, they are one-hop neighbors.

© 2005 by Chapman & Hall/CRC

1.7 Technology Needed

Less research has been done on the fundamental mathematical/computational problems that need to be solved in order to provide a systematic approach to DSN system design. Issues of major interest include the optimal distribution of sensors, tradeoffs between the communication bandwidth and storage and communication, and the maximization of system reliability and flexibility. There are also a number of problems pertaining to communications that need to be resolved, e.g. the problem of collecting data from a large number of nodes and passing it to a specific node. The related problems are congestion at the collecting point and redundancy in the data obtained from the different nodes.

Current areas of research should include (but not be limited to) the following topics:

1. A new robust spatial integration model from descriptions of sensors must be developed. This includes the problem of fault-tolerant integration of information from multiple sensors, mapping and modeling the environment space, and task-level complexity issues of the computational model. Techniques of abstracting data from the environment space on to the information space must be explored for various integration models.
2. A new theory of complexity of processes for sensor integration in distributed environments must be developed. The problem of designing an optimal network and detecting multiple objects has been shown to be computationally intractable. The literature gives some approximate algorithms that may be employed for practical applications. It has been shown that the detection time without preprocessing is at most quadratic for sequential algorithms. What is needed is further work based on these foundations for the computational aspects of more complex detection systems, not only in terms of algorithms for detection, but also for system synthesis.
3. Distributed image reconstruction procedures must be developed for displaying multiple source locations as an energy intensity map.
4. Distributed state estimation algorithms for defense and strategic applications must be developed (e.g. low-altitude surveillance, multiple target tracking in a dense threat environment, etc.).
5. A distributed operating system kernel for efficient synthesis must be developed.
6. DSNs are scalable, extensible, and more complex. Owing to the deployment of multiple sensors, each of which captures the same set of features of an environment with different fidelity, there is an inherent redundancy built into a DSN. The sensors complement each other, and the union of data obtained can be used to study events that would otherwise be impossible to perceive using a single sensor.

1.8 Contrast with Traditional Computing Systems

DSNs are quite different from traditional networks in a number of ways. Traditional computer and communication networks have a fixed topology and number of nodes. They are designed to maximize the rate of data throughput. In contrast, the purpose of DSNs is to sense, detect, and identify various parameters in unknown environments. The nodes consist of one of more sensors, of the same or different types, along with an associated processor and a transceiver [9].

The deployment of sensor nodes is also unique and differs with the applications. The sensors may be placed manually around the perimeter of a manufacturing plant or placed inside a building. They may be rapidly deployed by a reconnaissance team near the paths of their travel, or even randomly distributed out of a vehicle or an airplane over enemy terrain.

We believe that the vision of many researchers to create smart environments through the deployment of thousands of sensors, each with a short-range wireless communications channel and capable of detecting ambient conditions such as temperature, movement, sound, light, or the presence of certain objects, is not too far away.

© 2005 by Chapman & Hall/CRC

References

[1] Pradhan, S.S., Kusuma, J., and Ramchandra, K., Distributed compression in a dense sensor network, *IEEE Signal Processing Magazine*, 19, 2002.

[2] Schurgers, C., Kulkarni, G., and Srivastava, M.B., Distributed on-demand address assignment in wireless sensor networks, *IEEE Transactions on Parallel and Distributed Systems*, 13, 2002.

[3] Iyengar, S.S., Kashyap, R.L., and Madan, R.N., Distributed Sensor Networks—Introduction to the Special Section, *IEEE Transactions on Systems, Man, and Cybernetics*, 21, 1991.

[4] Iyengar, S.S., Chakrabaraty, K., Qi, H., Introduction to special issue on 'Distributed sensor networks for real-time systems with adaptive configuration,' *Journal of the Franklin Institute* 338, 651–653, 2001.

[5] Iyengar, S.S., and Kumar, S., Special Issue: Advances in Information Technology for High Performance and Computationally Intensive Distributed Sensor Networks, *Journal of High Performance Computing*, 16, 2002.

[6] Cullar, D., Wireless sensor networks, *MIT's Magazine of Innovation Technology Review*, February, 2003.

[7] Iyengar, S.S., and Brooks, R., Special Issue on Frontiers in Distributed Sensor Networks, *Journal of Parallel and Distributed Computing*, in press, 2004.

[8] Brooks, R.R., and Iyengar, S.S., *Multi-Sensor Fusion: Fundamental and Applications with Software*, Prentice-Hall, 1997.

[9] Chen, J.C., Yao, K., and Hudson, R.E., Source localization and beamforming, *IEEE Signal Processing Magazine*, 19, 30–39, 2002.

© 2005 by Chapman & Hall/CRC

2

Microsensor Applications

David Shepherd and Sri Kumar

2.1 Introduction

In recent years, the reliability of microsensors and the robustness of sensor networks have improved to the point that networks of microsensors have been deployed in large numbers for a variety of applications. The prevalence of localized networks and the ubiquity of the Internet have enabled automated and human-monitored sensing to be performed with an ease and expense acceptable to many commercial and government users. Some operational and technical issues remain unsolved, such as how to balance energy consumption against frequency of observation and node lifetime, the level of collaboration among the sensors, and distance from repeater units or reachback stations. However, the trend in sensor size is to become smaller, and at the same time the networks of sensors are becoming increasingly powerful. As a result, a wide range of applications use distributed microsensor networks for a variety of tasks, from battlefield surveillance and reconnaissance to environment monitoring and industrial controls.

2.2 Sensor Networks: Description

Sensor networks consist of multiple sensors, often multi-phenomenological and deployed in forward regions, containing or connected to processors and databases, with alerts exfiltrated for observation by human operators in rear areas or on the front lines. Sensor networks' configurations range from very flat, with few command nodes or exfiltration nodes, to hierarchical nets consisting of multiple networks layered according to operational or technical requirements. The topology of current sensor networks deployed in forward areas generally includes several or tens of nodes reporting to a single command node in a star topology, with multiple command nodes (which vary in number according to area coverage and operational needs) reporting to smaller numbers of data exfiltration points. Technological advances in recent years have enabled networks to ''flatten'': individual sensor nodes share information with each other and collaborate to improve detection probabilities while reducing the likelihood of false alarms [1,2]. Aside from the operational goal of increasing the likelihood of detection, another reason to flatten sensor networks is to reduce the likelihood of overloading command nodes or human operators, with numerous, spurious and even accurate detection signals. In addition, although a point of diminishing returns can be reached, increasing the amount of collaboration and processing performed among the nodes reduces the time to detect and classify targets, and saves power — the costs of

© 2005 by Chapman & Hall/CRC

communicating data, particularly for long distances, far outweighs the costs of processing it locally. Processing sophistication has enabled sensor networks in these configurations to scale to hundreds and thousands of nodes, with the expectation that sensor networks can easily scale to tens of thousands of nodes or more as required.

Sensor networks can be configured to sense a variety of target types. The networks themselves are mode-agnostic, enabling multiple types of sensor to be employed, depending on operational requirements. Phenomenologies of interest range over many parts of the electromagnetic spectrum, including infrared, ultraviolet, radar and visible-range radiations, and also include acoustic, seismic, and magnetic ranges. Organic materials can be sensed using biological sensors constructed of organic or inorganic materials. Infrared and ultraviolet sensors are generally used to sense heat. When used at night, they can provide startlingly clear depictions of the environment that are readily understandable by human operators. Radar can be used to detect motion, including motion as slight as heartbeats or the expansion and contraction of the chest due to breathing. More traditional motion detectors sense in the seismic mode, since the Earth is a remarkably good transmitter of shock waves. Acoustic and visible-range modes are widely used and readily understood — listening for people talking or motors running, using cameras to spot trespassers, etc. Traditional magnetic sensors can be used to detect large metal objects such vehicles or weapons, whereas they are unlikely to detect small objects that deflect the Earth's magnetic field only slightly. More recent sensors, such as those used in the past few decades, are able to detect small metal objects. They induce a current in nonmagnetic metal, with the response giving field sensitivity and direction.

Early generations of sensors functioned primarily as tripwires, alerting users to the presence of any and all interlopers or targets without defining the target in any way. Recent advances have enabled far more than mere presence detection, though. Target tracking can be performed effectively with sensors deployed as a three-dimensional field and covering a large geographic area. Sufficient geographic coverage can be accomplished by connecting several (or many) smaller sensor fields, but this raises problems of target handoff and network integration, as well as difficulties in the deployment of the sensors themselves. Deployment mechanisms currently in use include hand emplacement, air-dropping, unmanned air or ground vehicles, and cannon-firing. Effective tracking is further enhanced by target classification schemes. Single targets can be more fully identified through classification, either by checking local or distributed databases, or by utilizing reachback to access more powerful databases in rear locations. Furthermore, by permitting disambiguation, classification systems can enable multiple targets to be tracked as they simultaneously move through a single sensor field [3].

2.3 Sensor Network Applications, Part 1: Military Applications

Prior to the 1990s, warfighters planning for and engaging in battle focused their attentions on maneuvering and massing weapons *platforms*: ships at sea; tanks and infantry divisions and artillery on land; aircraft in the air. The goal was to bring large quantities of weapons platforms to bear on the enemy, to provide overwhelming momentum and firepower to guarantee success regardless of the quantity and quality of information about the enemy. Warfighting had been conducted this way for at least as long as technology has enabled the construction of weapons platforms, and probably before then also. In the 1990s, however, the United States Department of Defense (DoD) began to reorient fundamental thinking about warfighting. As a result of technological advances and changes in society as a whole, thinkers in the military began advocating a greater role for *networked* operations. While planners still emphasize bringing overwhelming force to bear, that force would no longer be employed by independent actors who controlled separate platforms and had only a vague understanding of overall battlefield situations or commanders' intentions. Instead, everyone involved in a conflict would be connected, physically via electronic communications, and cognitively by a comprehensive awareness and understanding of the many dimensions of a battle. Thanks to a mesh of sensors at the point of awareness and to computers at all levels of engagement and analysis, information about friendly and enemy firepower levels and situation awareness can be shared among appropriate parties. Information

© 2005 by Chapman & Hall/CRC

assumed a new place of prominence in warfighters' thinking as a result of this connectedness and information available to commanders. Even more comprehensive understandings of network-centric warfare also considered the ramifications of a completely information-based society, including how a fully networked society would enable operations to be fought using financial, logistical, and social relationships [4].

Critical to the success of network-centric warfare is gathering, analyzing, and sharing information, about the enemy and about friendly forces. Because sensors exist at the point of engagement and provide valuable information, they have begun to figure larger in the planning and conduct of warfare. Especially with the ease of packaging and deployment afforded by miniaturization of electronic components, sensors can provide physical or low-level data about the environment and opponents. They can be the eyes and ears of warfighters and simultaneously enable humans to remain out of harm's way. Sensors can provide information about force structures, equipment and personnel movement; they can provide replenishment and logistics data; they can be used for chemical or biological specimen detection; sensors can provide granular data points or smoothed information about opposition and friendly forces. In this way, microsensor data can be an important part of what defense department theorists term ''intelligence preparation of the battlespace''. Sensor data can be folded into an overall picture of the enemy and inform speculations of enemy intent or action.

2.3.1 Target Detection and Tracking

A primary application of networked microsensors is the detection and tracking of targets. The first modern sensor application and system was intended for this purpose. During the Viet Nam war in the 1960s, Secretary of Defense Robert McNamara wanted knowledge of North Vietnamese troop activity, and ordered the construction of an electronic anti-infiltration barrier below the Demilitarized Zone (DMZ), the line of demarcation between North and South Vietnam. The principal purpose of this ''McNamara Line'' would be to sound the alarm when the enemy crossed the barrier. The mission of the program was later changed to use sensors along the Ho Chi Minh Trail to detect North Vietnamese vehicle and personnel movements southward. Named Igloo White, the program deployed seismic and acoustic sensors by hand and by air-drop. Seismic intrusion detectors (SIDs) consisted of geophones to translate ground movements induced by footsteps (at ranges of up to 30 m) or by explosions into electrical signals. SIDs could be hand emplaced or attached to ground-penetrating spikes and air-dropped (air-delivered SIDs [ADSIDs; Figure 2.1]). Up to eight SIDs could transmit to one receiver unit over single frequency channels. All SID receiver units transmitted to U.S. Air Force patrol planes orbiting 24 h per day, with data displayed using self-contained receive and display (i.e. lamp) units or the aircraft's transceiver. Hand-emplaced versions of the SIDs weighed 7 lbs each and were contained in a $4.5'' \times 5'' \times 9''$ metal box. Smaller versions, called patrol SIDs (PSIDs), were intended to be carried by individual soldiers and came in sets of four sensors and one receiver. Each sensor weighed 1 lb, could be fitted into clothes pockets, and operated continuously for 8 h. Sensor alarms sounded by transmitting beeps, with one beep per number of the sensor [1–4]. Target tracking was performed by human operators listening for the alarms and gauging target speed and direction from the numbers and directions of alarm hits [5].

The Igloo White sensing systems required human operators to listen for detections, disambiguate noise from true detections, correlate acoustic and seismic signals, and transmit alarms to rear areas. It also employed a hub-and-spoke topology, with many sensors reporting to a single user or exfiltration point. More recent research into target detection and tracking has reduced or removed the requirement for human intervention in the detection and tracking loop. It has also proved the superiority of a mesh topology, where all sensors are peers, signal processing is performed collaboratively, and data are routed according to user need and location. Routing in a distributed sensor network is performed optimally using *diffusion* methods, with nodes signifying interests in certain types of data (i.e. about prospective targets) and supplying data if their data match interests published by other nodes. This data-centric routing eliminates the dependency on IP addresses or pre-set routes. The redundancy provided by

© 2005 by Chapman & Hall/CRC



Figure 2.1. An ADSID sensor from the Vietnam-era Igloo White program.

multiple sensor nodes eliminates single points of failure in the network and enables sensors to use multiple inputs for classification and to disambiguate targets. Reaching decisions at the lowest level possible also conserves bandwidth and power by minimizing longer range transmissions. If necessary (for operational reasons, power conservation, or other reasons), exfiltration is still often performed by higher power, longer range nodes flying overhead or contained in vehicles [6].

Collaborative target tracking using distributed nodes begins as nodes local to a target and these nodes cluster dynamically to share data about targets; the virtual cluster follows the target as it moves, drawing readings from whichever nodes are close to the target at any instant. One method of obtaining readings is the closest point of approach (CPA) method. Points of maximum signal amplitude are considered to correspond to the point of the target's closest physical proximity to the sensor. Spurious features, such as ambient or network noise, are eliminated by considering amplitudes within a space–time window and resolving the energy received within that window to a single maximum amplitude. The size of the window can be adjusted dynamically to ensure that signal strength within the window remains approximately constant to prevent unneeded processing. This can be achieved because it is unlikely that a ground target moves significantly enough to change signal strength appreciably. As the target moves through the sensor field, maximum readings from multiple sensors can be analyzed to determine target heading. As many sensors as possible (limited by radio range) should be used to compute the CPA, because too few sensors (fewer than four or five) provides an insufficient number of data points for accurate computations.

Improved tracker performance can be achieved by implementing an extended Kalman filter (EKF) or with the use of techniques such as least-squares linear regression and lateral inhibition. Kalman filters compute covariance matrixes that vary according to the size of the sensor field, even if the size varies dynamically to follow a moving target. Least squares is a straightforward method of approximating linear relations between observed data, with noise considered to be independent, white Gaussian noise [7]. This is an appropriate approximation for sensor data owing to the large sample size and large proportion of noise achieved with the use of a field of sensors. In lateral inhibition, nodes broadcast intentions to continue tracks to candidate nodes further along the target's current track, and then wait a period of time, the duration of which is proportional to how accurate they consider their track to be (based on the number and strength of past readings, for example). During this waiting period, they

© 2005 by Chapman & Hall/CRC

listen for messages from other nodes that state they are tracking the target more accurately. If other nodes broadcast superior tracks, then the first node ceases tracking the target; if no better track is identified, then the node continues the track. The performance of these tracking schemes has varied. This variability has shown that certain trackers and certain modalities are better suited to certain applications. For example, in tracking tests performed on sensor data obtained by researchers in the Defense Advanced Research Projects Agency (DARPA) Sensor Information Technology (SensIT) program at the U.S. Marine Corps base in Twentynine Palms, CA, trackers using EKFS produced slightly more accurate tracks, but lateral inhibition can more ably track targets that do not have a linear target trajectory (such as when traveling on a road) [3,8,9].

Another target-tracking algorithm using intelligent processing among distributed sensors is the information directed sensor querying (IDSQ) algorithm. IDSQ forms belief states about objects by combining existing sensor readings with new inputs and with estimates of target position. To estimate belief states (posterior distributions) derived from the current belief state and sensor positions, sensors use entropy and predefined dynamics models. IDSQ's goal is to update the belief states as efficiently as possible, by selecting the sensor that provides the greatest improvement to the belief state for the lowest cost (power, latency, processing cycles, etc.). The tradeoff between information utility and cost defines the objective function used to determine which nodes should be used in routing target information, as well as to select clusterhead nodes. The information utility metric is determined using the information theoretic measure of entropy, the Mahalanobis distance (the distance to the average normalized by the variation in each dimension measured) and expected posterior distribution. The use of expected posterior distribution is considered particularly applicable to targets not expected to maintain a certain heading, such as when following a road. The belief states are passed from leader node (clusterhead) to leader node, with leaders selected by nearby sensors in a predefined region. This enables other sensors to become the leader when the original leader fails, increasing network robustness [10].

2.3.2 Application 1: Artillery and Gunfire Localization

The utility of distributed microsensors for tracking and classification can be shown in applications such as gunfire localization and caliber classification. Artillery localization illuminates several issues involving sensing. Artillery shell impact or muzzle blasts can be located using seismic and acoustic sensors; however, the physics behind locating seismic or acoustic disturbances goes beyond mere triangulation based on one-time field disturbances. When artillery shells are fired from their launchers, the shells themselves do not undergo detonation. Instead, a process called ''deflagration'' occurs. Deflagration is the chemical reaction of a substance in which the reaction front advances into the unreacted substance (the warhead) at less than sonic velocity. This slow burn permits several shock waves to emanate from the muzzle, complicating the task of finding the muzzle blast. This difficulty is magnified by the multiple energy waves that result from acoustic reflections and energy absorption rates of nearby materials, and seismic variations caused by different degrees of material hardness. Furthermore, differences in atmospheric pressures and humidity levels can affect blast dispersion rates and directions. Causing additional difficulty is the presence of ambient acoustical events, such as car backfires and door slams in urban areas. Each of these factors hampers consistent impulse localization [11].

Whereas the sonic frequencies of artillery blasts are low enough to prevent much dispersion of the sounds of artillery fire, the sounds of handgun and rifle fire are greatly affected by local conditions. One distributed sensor system that tracks the source of gunfire is produced by Planning Systems, Incorporated. The company has tested systems consisting of tens of acoustic sensors networked to a single base station. The system employs acoustic sensors contained in $6'' \times 6'' \times 4''$ boxes that can be mounted on telephone poles and building walls, although sensors in recent generations of production are the size of a hearing aid [12]. When elevated above ground by 10 m or more using telephone poles approximately 50 m apart, sensors can locate gunshots to within 1–2 m. The system uses triangulation of gunshot reports to locate the source of the firings; at least five sensor readings are needed to locate targets accurately. Primary challenges in implementing this system include acoustic signal dispersion,

© 2005 by Chapman & Hall/CRC

due to shifting and strong winds, and elimination of transient acoustic events. To counter the effects of transient noise, engineers are designing adaptive algorithms to take into account local conditions. In the past these algorithms have been implemented on large arrays, but recent implementations have been performed on a fully distributed network. For use on one such distributed net, researchers at BBN Technologies have designed a parametric model of gunshot shock waves and muzzle blast space–time waveforms. When at least six distributed omnidirectional microphones are used, the gunshot model and waveforms can be inverted to determine bullet trajectory, speed and caliber. When a semi-permanent distributed system can be constructed, two four-element tetrahedral microphone arrays can be used. A three-coordinate location of the shooter can also be estimated if the muzzle blast can be measured.

The researchers have developed a wearable version of the gunshot localization system for the implementation with a fully distributed, *ad hoc* sensor network. Each user wears a helmet, mounted with 12 omnidirectional microphones, and a backpack with communications hardware and a global positioning system location device. The use of omnidirectional sensors eliminates the need for orientation sensors to determine the attitude of an array of directional sensors. The system detects low frequency (<10 kHz) gunshot sounds, and so is relatively impervious to waveform scattering and dispersion typical of higher frequency transmissions. It works primarily by detecting the supersonic shock wave made by the bullet as it travels through the air, thus avoiding a reliance on sensing muzzle blasts or other effects local to the shooter which can be masked or minimized. Its performance is enhanced if microphones can be located laterally compared with the bullet trajectory, to ease comparisons between the Mach cone angle (the cone produced by the edges of the expanding sound waves) and trajectory angles. Of course, this enhancement cannot be relied upon in a distributed system with microphones located on mobile users. If a spatially distributed system can be employed, then adequate muzzle and shock arrival time estimates can be obtained with less than 8 kHz bandwidth (20 kHz sampling). The 8 kHz bandwidth is also sufficient to classify bullet caliber. System tests in 1997 showed that the helmet-mounted system detected all 20 rounds fired, using 22, 30 and 50 caliber weapons. Some 90% of shots were determined to within 5° of azimuth, and 100% to within 20°. All shots were detected to 5° of elevation. Some 50% of shots were detected to be within 5% range accuracy, and 100% to within 20% of the actual range. Designers blamed the poor range-estimation performance on muzzle detection performance, attributable to algorithmic difficulties [13].

2.3.3 Application 2: Unmanned Aerial Vehicle Sensor Deployment

Distributed ground sensors without organic long-range exfiltration avenues do not have standoff capabilities. Their advantage lies in providing *in situ* sensing, or sensing done at or very near the location of the target. Sensing very near the target minimizes signal attenuation due to distance, atmospheric changes, radiation interference, or similar factors. On the other hand, *in situ* sensing suffers from challenges that standoff sensing avoids, such as an absence of robust power sources, narrowed or inflexible field of regard, and reliance on extended communications. While enabling many of its advantages, many difficulties of *in situ* sensing can be avoided or overcome through effective deployment of sensors, particularly through the use of unmanned aerial vehicles (UAVs). Present generations of smart sensors can be deployed via UAV quite effectively, because the sensors are self-aware and self-configuring. This eliminates the need for pre-established links, clear lines of sight, or defined routes around topological and connectivity disturbances, such as hills or buildings. Distributed sensors employ collaborative signal processing and mesh-style networking methods, such as diffusion routing, to guarantee message delivery despite uncertain connectivity, node failure, or variations in the number of nodes needed to determine target location or type. With this sort of intelligence built into its sensors, a UAV flown from a nearby ground station or with a preplanned route and instructions to drop sensors at waypoints can deploy a network of sensors with sufficient sensing power and longevity to perform numerous tracking tasks and to exfiltrate data. As UAV and sensor network capabilities improve, the need for preplanned routes, human operators, preset communications relays, and portage

© 2005 by Chapman & Hall/CRC

of redundant batteries will also fade. One result of all these improvements is that networked battlefields of the future will use UAVs heavily to scout enemy positions, track targets, and relay messages — as well as deliver sensors and exfiltrated sensor reports.

Through the use of multiple UAVs, tracking can be performed in a decentralized manner. This eliminates single points of failure, lessens power requirements on central nodes, and eliminates reliance on single, high-latency links. By using fusion algorithms to combine readings from multiple UAVs covering overlapping regions, all targets within the covered area can be mapped, with their headings and directions determined. In one multiple-UAV fusion algorithm, when multiple observations are reported, Bayes's theorem is used to provide position estimates through linear combinations of observed information states. One advantage of this system is its maintenance of full information states, because each node maintains its complete information state estimate and transmits only incremental information updates. This system suffers, however, from a lack of processing power at the nodes. In one test, this resulted in nodes being unable to merge inputs from other nodes consistently. As long as few targets exist, the multiple nodes do not have trouble maintaining a consistent understanding of the targets' locations. When more targets appear than can be processed by the system, nodes unable to clear their buffers show a small number of targets not common to the fused picture. Another difficulty was seen in inaccuracies in UAV location maintenance. Whereas a large number of UAV passes over the target range will smoothe this error to zero, location errors can be introduced for circumstances with few UAV passes [14].

UAVs are so new to the military that operational units do not currently use multiple, networked UAVs to sense in single regions. In fact, the current field-tested and operational UAVs organic to forward-deployed military units have limited flight and communications ranges and limited networking power. One such small UAV currently in predeployment testing and use by the U.S. Marine Corps is the Dragon Eye. Developed by the Marine Corps Systems Command in Quantico, VA, the Dragon Eye is a "backpackable" UAV. It has a 4 ft wingspan and weighs 5.5 lb itself; the ground station, required to fly the UAV and to communicate (radio frequency) with the UAV, adds an additional 7 lb and includes goggles for video viewing. The UAV's wings can be detached from the fuselage for transport. Its sensor package currently includes off-the-shelf color video and low-light black-and-white video cameras. Uncooled infrared cameras are also planned [15]. A UAV that is slightly larger than the Dragon Eye and is in use for maritime applications is produced by Insitu Group. The Seascan UAV is intended specifically for shipboard imaging, and thus can be used for littoral operations. Like the Dragon Eye, Seascan UAVs must be launched using a catapult or elastic to shoot the UAV into the air; recovery is done by landing the UAV in an area of flat ground nearby (in the case of the Dragon Eye) or into a shipboard recovery net (in the case of the Seascan). Current sensor packages (summer 2003) include a color video camera installed in the nose of the craft. The daylight camera has a 45° field of view, pan-tilt capability, and the capacity to remain locked on a target while the UAV maneuvers. Other sensors being built have included infrared cameras, LIDAR systems, air particle sensors, and hydrometers [16].

Like the sensors deployed in the Dragon Eye, sensors in the Seascan use a hub-and-spoke topology rather than the mesh that a fully networked system would employ. Likewise, the nodes do not collaborate to reach decisions about a target; instead, a single video camera reports to the base station user what it shows. Both the Seascan and the Dragon Eye employ a single UAV deployed by a single operator; no plans exist to use a network of Dragon Eyes. However, research is under way in swarming UAVs; these would be fully networked and capable of acting collaboratively. Future UAV-mounted sensor systems will report multimodal information, with multiple nodes collaborating to lower false alarm rates, verify target types, and perform other more sophisticated tasks. One direction research into UAVs is headed is into micro-air vehicles (MAVs). MAVs are only several inches in wingspan, with further reductions to insect-sized vehicles expected in the future. One MAV built by a group from the University of Florida at Gainesville and the NASA Langley Research Center relayed live video streams to ground users (Figure 2.2) [17]. However, tremendous challenges exist in enabling MAVs to sense over long periods of time and extended geographic areas, such as power requirements, communications distances to users and databases, and aerodynamic obstacles (e.g. flying in the presence of winds).

© 2005 by Chapman & Hall/CRC



Figure 2.2. MAV in flight, with video sensor feed transmitted to ground computer.

2.3.4 Target Classification

A significant advancement over previous detection and tracking applications is the capacity to classify targets rather than merely to detect them. Early sensing applications such as Igloo White, and even later applications such as REMBASS, were notorious for triggering alarms as a result of animal intrusions. Accurate classification of targets would reduce the probability of false alarm in each sensor and network, as well as provide the user with knowledge of target type and ideally even identify the precise target itself within the target type. This enables superior knowledge of the target, which in turn facilitates tracking over long distances and tracking in a multitarget, multisensor situation. In the latter case, a target such as a vehicle is identified by characteristics specific to that particular target, through distinctive engine noises (such as piston knocks), suspension irregularities (such as a squeaky spring), etc. This sort of identification is very helpful in solving the computation- and communications-intensive multiple-target tracking situations, when targets are not constrained to roads or other predetermined pathways.

Classification occurs through the comparison of unknown target signatures with so-called training data obtained from known target types. Data on targets is often reduced to feature vectors, or sets of data with the lowest dimensionality that can identify the target distinctly. Detection in the acoustic, seismic, infrared or magnetic modes occurs when feature vector energies exceed a predetermined threshold; classification occurs when the specifics of the feature vector match the signatures in the training data. Classification in the visual mode operates not through energy exceeding a threshold, but (in the case of unmanned classifiers) by matching signature data to the training data. This makes selection of training data and accuracy of test data obtained from the field critical. Yet field data are always obtained under less than ideal conditions. Factors such as variability in sensor location, instrumentation quality, and latency due to network configurations amplify other, less avoidable variations, such as Doppler effects due to target motion and small differences in target locations, stealthiness, and mechanical functioning [3]. However, despite these opportunities for uncertainty, small differences can be eliminated in the process of reducing data to feature vectors to reduce target dimensionality. To obtain feature vectors, target spectra of various modalities can be obtained efficiently through Fourier analysis of the data to yield power spectral densities, grouped with data points corresponding to frequencies of interest [8]. Given that classification involves more than the comparison of target energy levels to preset thresholds, consistent determination of the feature vectors from the full data set is important to ensure accurate classification. Each detection event yields a single

© 2005 by Chapman & Hall/CRC

feature vector; multiple detection events can be collapsed into a single, representative vector to save bandwidth.

Correlations and relationships between feature vectors can be determined using any of several signal-processing algorithms. Common algorithms include *k*-nearest neighbor, maximum likelihood, support vector machine, singular value decomposition (SVD), and principal component analysis. The *k*-nearest neighbor algorithms test fitness between training data and test vectors, and then combine classifications from the nearest *k* neighbors using algorithms such as majority vote to decide on a single vector. Maximum-likelihood classifiers model feature vectors from similar targets based on Gaussian density functions and classify the models through algorithms such as expectation-maximization. Support-vector-machine classifiers are learning machines (like neural networks) that can perform binary classification for pattern recognition and real-valued function approximation for regression estimation. SVD can infer relationships between data through the use of matrix decomposition of principal input values. In principal component analysis, the principal values are analyzed from feature vectors, broken down according to their most basic components, and grouped into regions where deviations from maxima and minima are most extreme (to retain the variation in the original data set as much as possible), and uncorrelated [18].

In distributed sensor networks, targets are classified using multiple measurements performed (sometimes multiple times) at multiple nodes. Fusion of data obtained in multiple modalities is also a beneficial method to classify targets, because most modalities are largely orthogonal, thereby increasing the likelihood that additional data will contribute significantly. This process is facilitated by nodes that have sensors from several modes colocated; commonly used modes include infrared, acoustic, seismic and magnetic. The multiple measurements, whether from a single node using multiple modes or from multiple nodes, are integrated by the network to arrive at a single result for each classification event. To arrive at this single "answer," all classifiers, regardless of application, use data fusion, decision fusion, or a combination of the two. When conducting data fusion, classifiers fuse feature vectors of all measurements prior to arriving at a final classification decision. In decision fusion, each classifier individually classifies targets and then sends this information to other nodes for final determination of target type [19]. Decision fusion is superior when target information is uncorrelated and statistically independent, enabling each node to arrive at a separate, independently valid decision. When data are correlated, data fusion is needed because separate nodes are likely to decide similar outcomes, making the fused decisions less than ideal. Fusion decisions must be based on observations that are as distinct from each other as possible to avoid skewed results. Decision fusion offers the advantage of minimizing communications burdens, because the messages passed among nodes are smaller (scalar yes/no classification results per event rather than multiple, high-dimensional vectors specifying target characteristics). Decision fusion also offers the benefit of minimizing computational burdens at the nodes and requiring less data to train nodes accurately, due to the reduced dimensionality of the transmitted data [20].

2.3.5 Application 3: Vehicle Classification

Performance statistics of a distributed classifier were gained from data taken during several experiments performed using a network of distributed microsensors deployed at the Marine Corps Air–Ground Combat Center (MCAGGC) in Twentynine Palms, CA. In each of these experiments, conducted from 2000 to 2002 by researchers from the DARPA SensIT program, microsensors deployed along roadways and near an intersection located and tracked vehicles using multimodality sensing, including at various times via acoustic, seismic and magnetic modes (Figure 2.3). The sensors were connected in an *ad hoc* network and reported the positions of the vehicles, their speed and direction of travel, and displayed the results to a base station located some kilometers away. In some cases, classification algorithms analyzed the seismic and acoustic data to determine vehicle type. In other cases, passive infrared sensor nodes detected the presence and motion of ground vehicles and triggered an imager, located in the middle of the sensor field, at the appropriate time. During a third type of experiment, 15 magnetometers were

© 2005 by Chapman & Hall/CRC

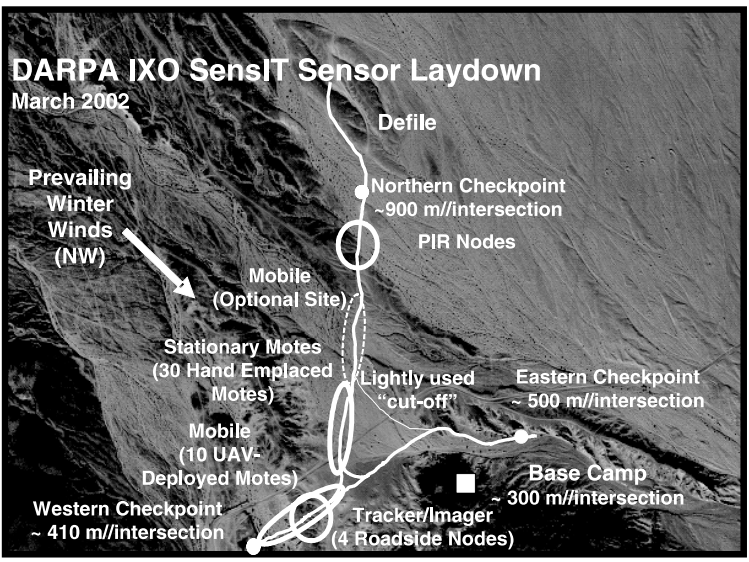


Figure 2.3. Deployment of sensors for the March 2002 SITEX experiments at Twentynine Palms MCAGCC, CA.

deployed on a road, by hand emplacement and UAV drop. The UAV returned to query the ground network from the air for vehicle detection, timestamp and speed, and then exfiltrated the responses to a remote base camp. Targets employed during the experiments included military vehicles (amphibious assault vehicles [AAVs], Dragon Wagons [DWs], HMMVW [HVs], and M1 tanks) and civilian vehicles (SUVs).

Numerical classification results are available from several of the SensIT experiments. One series of experiments, employing acoustic data taken from runs of AAVs, DWs, and HVs, illustrates the superiority of a classifier employing decision fusion and data fusion simultaneously. Local exchange of high-bandwidth feature vectors can improve measurement signal-to-noise ratios, with the results combined using a global exchange of low-bandwidth decisions across regions to stabilize signal fluctuations. In fact, reliable, high-level classification decisions can be reached through the use of fairly unreliable local decisions from multiple regions where signals are strongly coherent. Furthermore, the superiority of decision fusion utilizing local data fusion holds for networks regardless of whether their decisions are hard, soft or noisy (where hard fusion transmits ones or zeroes only, depending on the measured values, soft fusion employs real-valued measurements, and noisy fusion uses hard decisions transmitted over noisy and unreliable links). Research has shown that, under mild conditions for soft, hard and noisy fusion, the probability of misclassification approaches zero as the number of sensor reports from distinct but coherent regions increases. These results show that a network configuration currently preferred by planners of a distributed microsensor network, in which local sensors share high-bandwidth feature vectors but transmit to regional sensor clusters using low-bandwidth decisions, offers superior sensing performance: a relatively small number (under 50) of local decisions with a high probability of error (0.2 or 0.3) can be fused to yield a very reliable final decision (probability of error near 0.01) [21].

SensIT researchers also used data gathered during experiments at Twentynine Palms to analyze the efficiency of multimodality fusion based on SVD. Employing time series data from acoustic and seismic sensor readings, CPA data were determined to construct feature vectors for each target. Data within a time window of 4 to 5 s around the peak of each sample were used, to ensure accuracy of the CPA and eliminate superfluous noise. These data were divided into consecutive frames of 512 data points sampled at 5 kHz (0.5 s each), with a 0.07 s overlap with each of their neighbors. Power spectral densities of each frame were stored as a column vector of 513 data points corresponding to frequencies from 0 to

© 2005 by Chapman & Hall/CRC

Table 2.1. Classifier results. Data from SensIT experiment, 2000

Actual vehicle	Sensed vehicle			Classified correctly (%)
	AAV	DW	HV	
AAV	117	4	7	94
DW	0	106	2	98
HV	0	7	117	94

512 Hz. Using an eigenvalue analysis, the algorithm computed an unknown sample vector's distance from the feature space of a database of targets. The distance between the sample vector and the target database is considered the likelihood that the sampled frame corresponds to the frame of the target in the database, where the databases are grouped by attribute. The algorithm was employed to classify the three target vehicles (i.e. AAVs, DWs and HVs), with CPA peaks detected by hand rather than by the algorithm. Only one vehicle was driven through the sensor field at a time, under environmentally noisy conditions (particularly wind). As shown in Table 2.1, the algorithm classified vehicles correctly roughly 95% of the time, even under suboptimal conditions [9].

2.3.6 Application 4: Imaging-Based Classification and Identification

Target classification based on target images (still or moving) offers significant advantages over classification based on other sensing modalities. Whereas sensor fields operating in seismic, acoustic and magnetic modes can detect and track targets readily, the ability to discriminate between targets using these modes is not as simple. For instance, many vehicles sound similar or have similar magnetic signatures, and distinguishing personnel by the sounds or impacts of their footfalls is very difficult. Using video to classify targets, though, can provide more guaranteed results due to the rich feature set available with imagery. There is a large variability between target classes apparent in the visual mode, and a large number of target-specific features per class. For personnel identification, visually distinct characteristics, such as facial features and gait peculiarities, provide distinct signatures that are difficult to mimic. In the case of inorganic targets such as vehicles, identifying marks such as tire types, rust-spot locations, canvas versus hardtop distinctions and the like make targets readily identifiable. Imagery also makes a particularly compelling classification application because people rely so much on what their eyes provide — in the case of target classification, "seeing is believing." When connected to a large database of target identities, microsensor networks with video imaging systems can permit a field commander to ascertain his target's identity and be confident of the results of his mission. This capability also permits tracking systems to track multiple, similar targets. The primary drawback of an image-based classification system, though, is that the transmission of images occupies significant bandwidth. In a low-power, small-bandwidth system such as a microsensor network, this has prevented imagers from being employed on a scale similar that of the other modalities. However, as algorithms for feature extraction and image compression improve, and as camera sizes continue to diminish, microsensor networks that classify and identify targets using images will increase in utility and prominence.

Visual-mode and infrared-mode images are useful to imaging-classification systems. Uncooled microbolometer thermal imaging sensors can classify vehicles, for instance, according to temperature profiles based on normalized histograms or statistical properties of target pixel luminances compared with background information. This permits the classification of targets in daytime or nighttime, and through obscurants such as smoke, fog, or changes in lighting. Both infrared- and visual-mode classifiers have historically worked by comparing unknown target signatures with templates stored in a database. A disadvantage of this system is that templates must be available for multiple target orientations, as well as for each target. Shape-based classifiers compare features of targets with training

© 2005 by Chapman & Hall/CRC

data, using categories such as area and perimeter (both of which depend on range to target), orientation, spread (how unevenly an object's mass is distributed about the centroid of the target), elongation (the degree to which an object's mass is concentrated along the target's major axis), and compactness (which measures the complexity of the target shape) according to the formula:

$$\text{Compactness} = \frac{\text{Area}}{(\text{Perimeter})^2}$$

Comparisons between template and targets may be made using any of a number of decision-making algorithms, such as decision trees, neural networks, and Bayesian networks [22].

An alternative classifier compares target features rather than shapes, and offers the advantage of operating when the target is partially occluded. In one example of an image classifier, moving targets are illuminated first by eliminating the background using subtraction. Because background images are not static, pixels from the background are modeled as a mixture of Gaussians, or in the nonparametric case the actual probability density functions (pdfs) of the pixel intensities are used. The Gaussians or pdfs are compared with pixels of the target image, with only pixel intensities above a threshold considered to belong to the target. Next, depending on the proximity of the cameras detecting the target, a correlation of pixel neighborhood intensities is performed. For systems with cameras located far apart, a scale-invariant feature comparison is done. This is done using a scale-space approach, in which feature points that can be detected consistently in images from different viewpoints, i.e. points that vary in only two dimensions, are compared. Invariance is achieved further by obtaining rotation-invariant feature descriptors, consisting of vectors of 15 elements corresponding to higher order derivatives of the image. These descriptors can be chosen to isolate only certain, readily identifiable elements of the target. They can then be compared by obtaining the minimum Mahalanobis distance between each pair of descriptors, to enable target classification and identification [23].

Another imaging classifier uses distributed imaging sensors that have 360° cameras with software to piece together views that do not have the distorted, fish-eye look typically associated with panoramic cameras. This "normal view" can be achieved by projecting a viewplane at the angle and distance desired from the viewer to display an appropriate image; pixels from the panoramic image that show up on the viewplane behind the image are recorded and used to construct the usual perspective view. As a result, any number of views can be constructed from the same panoramic source. Because they operate independent of camera orientation, these sensing systems lend themselves to microsensor applications in which the sensor nodes are remotely deployed, such as by UAV or cannon-launch. One prototype vision node contains a 180° fish-eye lens and a 12 bit, 1 megapixel, 30 frames-per-second CCD color camera. The visual and infrared cameras are housed in a 2.5 cm × 2.5 cm × 1 cm unit. An alternate arrangement uses eight "regular" cameras, each with a 55° field of view, arranged in an octagon such that their fields of view overlap and provide a 360° view. One commercial camera suitable for this arrangement occupies less than 1/3 cm^3 and has a built-in lens and active pixel focal plane. Software merges the views and permits pan, tilt, and zoom capabilities.

Image and video transfer consumes considerable energy, whereas sensing in acoustic, seismic or magnetic modes does not. A power-efficient system, therefore, uses low-bandwidth sensors to cue an imager to power up only when a target comes into view. In one scenario envisioned by the designers, the energy requirements of detecting and classifying tanks and heavy trucks are determined. Hand-emplaced acoustic and seismic sensors detect targets, and cue an imager. Designers assumed a traffic usage of 200 vehicles (20 of them trucks) per hour during the day, and 30 vehicles/5 heavy trucks per hour at night. A target detection report consumes 200 bits, and an image chip consumes 10 kilobits (images sent on request). The camera takes approximately 20 still frames of an intersection and exfiltrates the images via satellite. A breakdown of system power requirements is provided in Table 2.2. Employing this usage model and these performance statistics, a 9 V battery would last 65 days; and it would last 34 days after taking 100 images per day and exfiltrating them via Iridium to a classification

© 2005 by Chapman & Hall/CRC

Table 2.2. Power requirements of a sensor system with satellite image exfiltration

	Average daytime power	Energy/day (J)
Data acquisition	350 μW	20
Image processing	5 mW	300
Comms (target report)	360 μW	23
Comms (image chips)	3 J/target	n/a

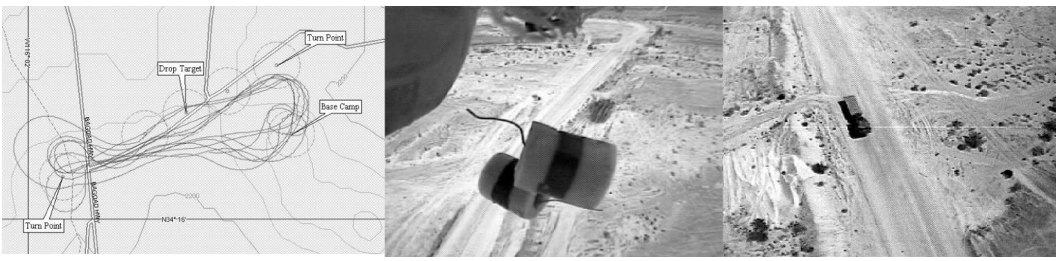


Figure 2.4. A UAV magnetometer drop: the racetrack pattern flown by a UAV; image from the UAV nose of foam-wrapped magnetometer nodes being dropped; image of a DW taken from the UAV.

algorithm in the rear, which compares the target images with a database of templates and permits viewing by analysts [24].

Experiments and real deployments in recent years have shown the practicality and utility of image-based tracking and classification. During one system test and demonstration in March, 2001, performed at MCAGCC in Twentynine Palms, CA, researchers from the University of California at Berkeley hand-launched a UAV to deploy sensors, record target images, and transmit the images to a base camp. A UAV carrying eight nodes with magnetometers flew a preset pattern over a roadway intersection. According to drop points loaded into the UAV software, the craft deployed half the magnetometers on a road; seven other the nodes were hand emplaced along the road. Targets included DWs, HVs and vehicles of opportunity (civilian and military vehicles). After deploying the nodes and waiting for targets to traverse the field, the UAV returned to fly over the deployed nodes and query the ground network for vehicle detection, time of occurrence, and vehicle speed. It exfiltrated the responses to a remote base camp and relayed images taken from cameras mounted in its nose and body. Although the system did not classify targets, it demonstrated the practicality of a full remote-deployment sensor–imager system. Classification software could be installed on the UAV or on the nodes with little difficulty to make this a truly distributed classification system.

2.4 Sensor Network Applications, Part 2: Civilian Applications

Networked microsensors can be used for a host of applications other than detection, tracking, and classification of military targets. Sensors' small size, the flexibility and robustness provided by *ad hoc* networks, and the variety of ways sensor data can be reported to users combine to ensure that sensor nets can be adapted to a variety of applications. Although privacy concerns have been raised due to the impending ubiquity of interconnected, distributed microsensor networks (especially considering visual sensing), the possibilities of distributed and remote sensing seem endless [25]. Some "civilian" applications of sensor networks include home security illegal-entry sensing, industrial machinery wear sensing, traffic control sensing, climate monitoring, long-term *in situ* medical monitoring of people and

© 2005 by Chapman & Hall/CRC

animals, agricultural crop monitoring (both pre- and post-harvest *in situ* and remote sensing), aircraft control-surface embedded sensing, and so on. Two such applications that have been tested outside of the laboratory include aiding the recovery of endangered plant species and personnel/heartbeat sensing.

2.4.1 Application 5: Monitoring Rare and Endangered Species

Owing to their capacity to provide data inexpensively over long periods of time and large geographic areas, sensor networks can be used to monitor biological and environmental conditions that would tax the resources of human monitors, such as in the case of temperature or rainfall extremes.

One such application is providing longitudinal environmental data at sites with endangered plants and animals, to enable analysis of environmental effects on the species. Data helpful in determining habitat conditions include temperature, humidity, rainfall, wind and solar radiation. The same information recorded for areas that do not contain the endangered species enables scientists to infer the role of climate in the species' distributions. The presence of endangered species adds an additional dimension to this use of sensor networks, though: the sensors themselves must not disturb the environment. In particular, the sensors must require little maintenance to ensure that technicians and researchers disturb the species' habitats as little as possible. This effort not to disturb the environment is amplified in deployments of distributed and remote sensing where the environment itself attracts visitors. The sensors themselves must attract as little attention as possible to minimize degradation of visitors' experiences.

In one such instance of these demanding requirements, researchers installed a network of sensors at the Hawaii Volcanoes National Park, to measure climate data for studies of environmental effects on the development of the endangered plant species *Silene hawaiiensis*. Some locations of Hawaii have steep environmental gradients due to the presence of extremely varied terrain and weather conditions. This enables researchers to measure the impacts of a large variety of climates on a relatively small region. In 2000 and 2001, to monitor environmental effects on phenological events, such as periods of flowering and seed-set over long periods of time, environmentalists and park officials installed a distributed network of wireless microsensors. Modalities sensed included temperature, light, wind and relative humidity. About 100 sensors, radiating in the 900 MHz band, were deployed, to connect to an Internet link approximately 2 km from the rare plants. Ranges between sensors varied according to the sensors' placement above the ground: 10 cm of height afforded approximately 30 m of range, and a range of 100 m could be attained when sensors were placed 2 m above the ground (in trees). Power was provided by four "C-size" batteries per sensor, enabling the network to operate for about 6 weeks when sensors turned on briefly only every 10 min to perform sensing and communication functions, including transmission of images from the base station to an off-island data repository. (The team is investigating using alternate energy sources, such as solar and wind power.) Researchers developed a power-efficient routing scheme, i.e. multi-path on-demand routing, based on node reachability rather than node positions, and also used a routing scheme similar to directed diffusion. For data flow and visualization, the team designed a data-reduction scheme such that nodes could store the data locally. This includes using an "exception reporting" scheme in which values are reported only when they fall outside of a normal range determined at the beginning of the sensing (a model for the environmental results was constructed from early data). Data visualization was aided by reporting the data using color-coded icons of the individual sensors on a map, with red representing values outside of the norm and green representing values inside it.

The most innovative aspect of the sensing application was in the packaging and deployment of the sensors themselves. The desire not to impact visitors' experiences at the parked researchers to disguise the sensors by deploying them in "fake rocks" and short, hollow, tree-like tubes (Figure 2.5). Microfiber filler (or "Bondo") and plaster of Paris molds were used to create a variety of rock-like enclosures. For the tree branches, researchers used PVC pipe, with both the "branches" and the "rocks" painted to blend into the environment. The sensors themselves were packaged in plastic bags inside the enclosures,

© 2005 by Chapman & Hall/CRC



Figure 2.5. Fake "rocks" and PVC "tree limb" sensor enclosures.

to prevent weather damage. These sensor housings were transparent to the radio emissions of the network, and went largely undetected by park visitors [26].

2.4.2 Application 6: Personnel Heartbeat Detection

Advances in technology have enabled sensors to be sensitive enough to detect tiny changes in the environment. One application that capitalizes on such sensitivity is the use of radar to detect motionless people. This is possible because very sensitive radars detect infinitesimal environmental changes, including the shock waves sent through the human thorax produced by the beating heat or the expansion and contraction of the chest due to breathing. The motion of a person's chest due to breathing is easier for a radar sensor to detect — radar returns are typically ten times higher for people breathing than for the motion due to a beating heart. However, one radar sensor measured changes as small as three heartbeats per minute; the sensor was used in the late 1990s to measure the heart rates of Olympic archers and riflemen to determine whether their training permitted them to avoid the approximately five milliradian movement of the bow or rifle due to their heartbeat at the time of firing. A "flashlight" version of the radar has been mounted on a tripod for long-term remote sensing; the beam can be narrowed to 16° to allow directed sensing [27]. Other radars that can sense through foliage and nonmetal walls have also been produced recently. In one preproduction unit, manufactured by Advantaca, the beamwidths can be set to 90°, 120°, or 360°. They have a maximum range of 20 m and a minimum range of 0.01 m, making them ideal to sense the presence of humans in closed rooms. A visual liquid-crystal display is provided on a separate unit connected to the radar emitter via wire. The sensors can detect people through 4 ft of rubble (2 ft of which is considered void), and through reinforced concrete walls 1.6 ft thick. In the latter case, the subject was 10 ft from the radar, lying prone. The Advantaca personnel detection radars were used immediately following the attacks on September 11, 2001, to detect people in the rubble of the World Trade Center [28].

The short-range radars have been packaged into "micropower impulse radars" and networked for surveillance of larger areas. They can be dropped from UAVs or hand emplaced to cover areas of interest, with signals exfiltrated via satellite. The radars are packaged into nodes each weighing 21 oz and about the size of a tennis ball can, with a similarly shaped but smaller unit inside it. The internal unit rotates freely along its long axis to ensure that the antenna always points upright upon landing. A detection radius of 20–30 m per node can be achieved; in tests, a single UAV pass can drop enough nodes to cover a 16 km^2 area. The primary technology enabler for a network of microradars is ultrawideband communications. Ultrawideband has emerged in recent years as a technology important to many communications arenas, such as personal area communications, tagging, ranging, localization, and motion sensing. It uses very short pulses of energy spread over very wide bandwidths. It operates at very low duty cycle and short pulse length and so requires little power. It can operate at 100 Mbps or more, with each pulse covering multi-gigahertz of spectrum [29]. For sensing applications,

© 2005 by Chapman & Hall/CRC

ultrawideband microradars can sense the presence of people, vehicles, animals, or any human-sized target that moves. Advantaca personnel envision the radars being useful for many commercial applications beyond heartbeat sensing. Other applications considered include vehicle/traffic sensing, security and personnel sensing, and anti-collision sensing, as well as imaging systems such as baggage detection, roadway and runway inspection, and piping and infrastructure detection.

2.5 Conclusion

Recent years have witnessed tremendous growth in the capabilities of networked sensors. As a result, people are finding more and more applications for them. Applications cover a spectrum of uses: short-term to long-term, perimeter security to surveillance in depth, trip-wire alerts to target identification using database access. Early funding for sensor networks was provided primarily by the U.S. Department of Defense, so the applications in which the most development has been performed are military ones. Area surveillance, target detection, target tracking, and remote deployment mechanisms have all been under development for at least several years to decades. Progress has been made to the point that miniaturized, long-lived, distributed sensors can be deployed to sense in multiple modalities and report results to rear areas fairly consistently. These sensor networks can perform a variety of functions that are ultimately useful to both the military and civilian worlds. As a result, more and more applications useful to the home and to industry are beginning to appear. For instance, weather and environmental data are being sought by scientists in regions heretofore not measured [30], and industry advocates see uses for sensors in both the production of goods and their consumption. Distributed sensors are a link between the phenomenological world all around us and the increasingly computerized world of databases, processors, and analysis; hence, the data and information provided by sensors will only become more useful as time goes on, and the number of applications will continue to grow.

References

[1] Kumar, S., *et al.*, eds., Collaborative Information Processing Special Issue, *IEEE Signal Processing Magazine*, 19(2), 2002.

[2] Gharavi, H., and Kumar, S.P. eds., Special Issue on Sensor Networks and Applications, *Proceedings of the IEEE*, 19(8), 2003.

[3] Li, D., *et al.*, Detection, classification, and tracking of targets, *IEEE Signal Processing Magazine*, 19, 17–29, 2002.

[4] Cebrowski, A., and Garstka, J., Network-centric warfare: its origins and future, *Naval Institute Proceedings*, January, 28–35, 1998.

[5] Jeppeson, C., Acoubuoy, SpikeBuoy, Muscle Shoals and Igloo White, http://home.att.net/~c.jeppeson/igloo_white.html, 1999.

[6] Estrin, D., and Pottie, G., Directed diffusion: a scalable and robust communication paradigm for sensor networks, In *Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking (MobiCOM '00)*, Boston, MA, August 2000.

[7] Brooks, R., and Iyengar, S.S., *Mutli-Sensor Fusion: Fundamentals and Applications with Software*. Prentice Hall, 1998.

[8] Brooks, R., *et al.*, Distributed target classification and tracking in sensor networks, *Proceedings of the IEEE*, 91, 1163–1171, 2003.

[9] Freidlander, D., *et al.*, Dynamic agent classification and tracking using an *ad hoc* mobile acoustic sensor network, *EURASIP Journal on Applied Signal Processing*, 4, 371–377, 2003.

[10] Zhao, F., *et al.*, Information-driven dynamic sensor collaboration, *IEEE Signal Processing Magazine*, 19, 61–72, 2002.

[11] Swanson, D., Artillery localization using networked wireless ground sensors. Unattended Ground Sensor Technologies and Applications, *Proceedings of the SPIE*, 4743, 73–79, 2002.

[12] Lewis, G., Planning Systems Incorporated. www.PlanningSystemsInc.com.

© 2005 by Chapman & Hall/CRC

[13] Duckworth, G., *et al.*, Fixed and wearable acoustic counter-sniper systems for law enforcement. Sensors, C3I, Information, and Training Technologies for Law Enforcement. *Proceedings of the SPIE*, 2577, 1998.
[14] Ridley, M., *et al.*, Decentralized ground target tracking with heterogeneous sensing nodes on multiple UAVs, *Information Processing in Sensor Networks*, Zhao, F., and Guibas, L., eds., Springer-Verlag, Berlin, 545–565, 2003.
[15] Adams, C., Minidrones: near term ..., *Avionics Magazine*, November, 2002. www.avaiation today.com.
[16] www.insitugroup.com
[17] Ettinger, S.M., *et al.*, Towards mission-capable micro air vehicles: vision-guided flight stability and control, *Advance Robotics*, in press.
[18] Jolliffe, I.T. *Principal Component Analysis*. Springer-Verlag, 1986.
[19] Kokar, M., *et al.*, Data vs. decision fusion in the category theory framework, In *Proceedings of Fusion 2001 Fourth International Conference on Information Fusion*, vol. 1, 2001.
[20] D'Costa, A., and Sayeed, A., Collaborative signal processing for distributed classification in sensor networks, In *Information Processing in Sensor Networks*, Zhao, F., and Guibas, L. eds., Springer-Verlag, Berlin, 2003, 193–208.
[21] D'Costa, A., *et al.*, Distributed classification of Gaussian space–time sources in wireless sensor networks, submitted to *IEEE Journal on Selected Areas of Communications*.
[22] Thomas, R., and Porter, R. Omnisense© visually enhanced tracking system, Unattended Ground Sensor Technologies and Applications, *Proceedings of the SPIE*, 4743, 129–140, 2002.
[23] Pahalawatta, P., *et al.*, Detection, classification, and collaborative tracking of multiple targets using video sensors, In *Information Processing in Sensor Networks*, Zhao, F., and Guibas, L., eds., Springer-Verlag, Berlin, 2003, 529–544.
[24] Boettcher, P., and Shaw, G. Energy-constrained collaborative processing for target detection, tracking and geolocation, In *Information Processing in Sensor Networks*, Zhao, F., and Guibas, L., eds., Springer-Verlag Berlin, 2003, 254–268.
[25] Shepherd, D., Networked microsensors and the end of the world as we know IT, *IEEE Technology and Society Magazine*, 22, 16–22, 2003.
[26] Biagioni, E., and Bridges, K., The application of remote sensor technology to assist the recovery of rare and endangered species, *The International Journal of High Performance Computing Applications*, 16, 112–121, 2002.
[27] Greneker, E. Radar flashlight for through-the-wall detection of humans, presented at Aerosense97.
[28] http://www.advantaca.com
[29] Jones, E. Ultrawideband squeezes, In *MIT Technology Review*, September, 71–79, 2002.
[30] Lundquist, J., *et al.*, Meteorology and hydrology in Yosemite National Park: a sensor network application, In *Information Processing in Sensor Networks*, Zhao, F., and Guibas, L., eds., Springer-Verlag, Berlin, 2003, 518–528.

© 2005 by Chapman & Hall/CRC

3

A Taxonomy of Distributed Sensor Networks*

Shivakumar Sastry and S.S. Iyengar

3.1 Introduction

A distributed sensor network (DSN) consists of a set of sensors that are interconnected by a communication network. The sensors are deeply embedded devices that are integrated with a physical environment and are capable of acquiring signals, processing the signals, communicating and performing simple computational tasks. The sensors are deployed in various environments and the data gathered by sensors must be integrated to synthesize new information. Often, this synthesis must be performed reliably, and within fixed time limits to support business objectives. In certain applications, such as automation systems, these tasks must be performed periodically while satisfying demanding performance constraints.

The efficient synthesis of information from noisy and possibly faulty signals emanating from sensors requires the solution of problems relating to (a) the architecture and fault tolerance of the DSN, (b) the proper synchronization of sensor signals, and (c) the integration of information to keep the communication and computation demands low. From a system perspective, once deployed, a DSN must organize itself, adapt to changes in the environment and nodes, and continue to function reliably. Current technology trends and devices facilitate several architectures for DSNs. In this paper, we propose a taxonomy for DSN architectures. Such a taxonomy is useful for understanding the evolution of a DSN and for planning future research.

3.2 Benefits and Limitations of DSNs

A DSN is an evolution of a traditional approach that is used to acquire inputs from a collection of sensors to support user decision-making. The traditional approach to solving this problem is depicted in Figure 3.1(a). Data from a collection of sensors are gathered by connecting the sensors to interface cards in a computing system. These data are presented to applications in suitable formats, and applications present information that is synthesized from such data to users. Figure 3.1(b) shows the

*Preliminary portion of this paper is published in *Sensor Processing Letters*, April 2004.

© 2005 by Chapman & Hall/CRC

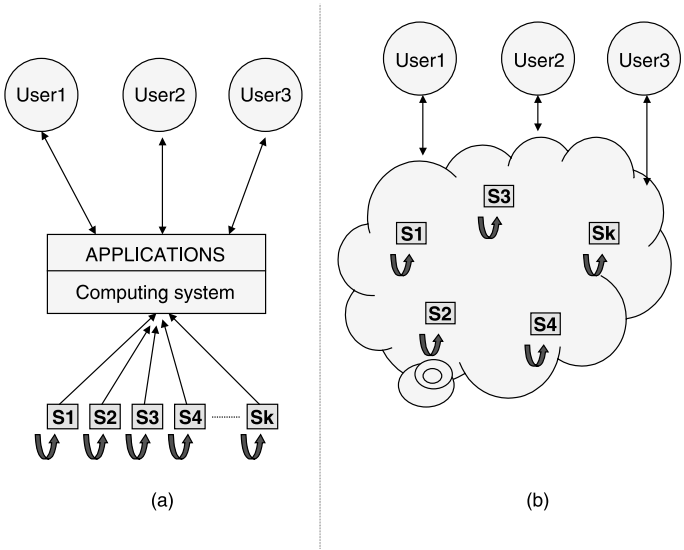


Figure 3.1. Fundamental architectural change in DSN.

organization of a DSN. Sensors are networked via wired or wireless media. There is no central computer that performs the coordination tasks. The network itself is a computer, and users interact with this network directly, possibly in interactive or proactive paradigms [1]. The data gathered by various sensors are integrated to synthesize new information using data fusion techniques [2].

DSNs may be distributed spatially and temporally to suit application demands. Often such networks lead to low-cost and reliable implementations. Quick response times are feasible for demanding sensing-loops. DSNs operate over large time scales and may be developed incrementally. Sensors can detect multiple input modalities, and combining such values provides new information that cannot be sensed directly. The overall throughput for sensing is increased because of the concurrent operations. The reliability of sensing is improved by using redundant sensors. Redundant sensors also make it feasible to develop fault-tolerant systems that degrade gracefully under exceptional circumstances. Groups of sensors work in complementary, competitive, or collaborative modes, and it is necessary to use different data fusion techniques to synthesize information in each of these cases.

If the individual nodes of a DSN require configuration or programming, such tasks are difficult because of scale. Communication mechanisms have to match application demands to achieve effective coordination between sensors and hence these mechanisms tend to be application specific. Sensors, typically, do not have individual identifiers. Distributed security mechanisms are still not mature and time synchronization across nodes is a significant challenge. New operating systems, communication protocols and security mechanisms are required to work with DSNs.

3.3 General Technology Trends Affecting DSNs

Following Agre *et al.* [3], we examine technology trends that impact the architecture of a DSN. A DSN is built using a large number of resource-constrained devices that are capable of acquiring inputs from the physical environment, processing and digitizing the data, communicating, and maintaining system cohesion. The primary impetus for distribution comes from an overruling of Grosch's law in 1970.

3.3.1 Grosch's Law Overruled

Grosch's law states that *the cost per machine instruction executed is inversely proportional to the square of the size of the machine*. This trend is now reversed [4–8]. Microprocessors that are embedded in sensors

© 2005 by Chapman & Hall/CRC

are relevant for DSNs. Network operating systems and tools for designing, configuring, and maintaining DSNs are coming into existence and promise to unleash the power of the aggregate.

The increased capabilities of individual sensor devices is consistent with Moore's law.

3.3.2 Moore's Law

Computational hardware performance doubles, for about the same cost, every 2 years. This trend of increased computational power is largely due to the mass production of microprocessors. At one time it may have been cost effective to optimize hardware resources, but the current trend is to favor sacrificing hardware utilization if the software development effort can be streamlined. Moore's law also motivates hardware realizations of critical functionality.

3.3.3 Greater "Siliconization"

Communications devices are manufactured with an increasingly larger proportion of integrated circuit (IC) technology (not necessarily limited to silicon, e.g. gallium arsenide). Thus, Moore's law is applicable to communications as well, although other factors (e.g. Shannon's law) may limit the rate of the curve compared with microprocessor development. Greater siliconization is caused by two effects: first, cost efficiencies result from the mass production in chip foundries; second, achieving greater transistor densities means that sensing, communicating, and computing capabilities can be integrated in a single chip, thereby further decreasing the costs of interconnection, especially by using wireless media.

Much of the functionality of a sensor may be more cost-effectively produced using CMOS and related technology. Power devices are also seeing potential realization in silicon by means of micro-electro-mechanical systems technology. Devices at the nanoscale are appearing on the horizon. Silicon sensors, i.e. analog and digital sensor devices that can be manufactured using chip-processing techniques, are becoming increasingly successful. We are witnessing a trend to host communication stacks and security mechanisms in silicon.

3.3.4 Increasing "Digitization"

Closely related to the above trend (of greater siliconization) is the trend to favor digital realizations over analog by collocating analog to/from digital conversion functions with sensors. For example, in smart sensors, a microprocessor or a digital signal processor can perform signal-processing functions that were previously performed either with additional analog circuits or in interface cards. Such collocation improves the reliability of data transmission and decreases costs (e.g. wiring, installation, maintenance).

3.3.5 Increasing Use of "Generic" Components

Start-up costs are taking an increasingly greater proportion of the overall cost (including production). The case of chip manufacturing is a good example of this general trend, since the majority of the cost is in manufacturing the first chip. Economy of scale then results from the fact that if more chips are produced, then the cost per chip is reduced.

This trend encourages the use of "generic" products (which may be over-qualified in some respects, but are still more cost effective) due to the economies of scale arising from its mass production. Other benefits, such as cost of deployment, maintenance, and training, also contribute towards reducing long-term operational costs.

3.3.6 Open Systems

In the current computer marketplace, there is a strong trend towards open systems. There are both economic and philosophical advantages to producing open systems, but there is also greater competition in any particular arena. In the context of a DSN, open systems appear to be unavoidable as

© 2005 by Chapman & Hall/CRC

generic intelligent sensors begin to flood the market. For easier integration with the physical world, DSNs must be able to operate with device-level open standards such as SERCOS and Ethernet; simultaneously, to integrate with supervisory systems, DSNs must also support interfaces to standards such as XML and SOAP.

3.4 Taxonomy of DSN Architectures

To realize systems that are depicted by Figure 3.1(b), we need to address the five major aspects that are shown in Figure 3.2. For each of the major aspects, there are variations that fundamentally alter the structure and performance of the DSN. These variations are captured in the taxonomy proposed in this chapter.

Following Agre *et al.* [3], we distinguish between function and implementation. *Function* refers to the basic operations or capability that the DSN aspect must address. *Implementation* refers to the methods that are used to accomplish the functions and the location of these methods. The location of the implementation is important because it is closely related to how the sensors are packaged. Packaging is a critical consideration because of costs. For each sensor, there is a certain minimum cost incurred in installing the sensor and maintaining it throughout the system lifecycle. By collocating groups of sensors, some of these costs can be amortized across sensors. While distributing sensors is desirable from an architecture perspective, collocating sensors is desirable from a cost perspective; hence, finding an appropriate balance between these considerations is the principal focus for packaging design.

The primary purpose of a DSN is to gather data from a physical environment within a predictable, bounded response time. A *hard-realtime* DSN is one in which the inability to respond to input events within their associated deadlines results in a system failure. The inability to meet the deadlines in a *soft-realtime* DSN does not result in a system failure, but a cost penalty is associated with tardiness. For example, if the data gathered are used to regulate the operation of a critical actuator (such as coolant in a power plant), then we need a hard-realtime DSN; in contrast, if the data gathered are used to locate a nearby restaurant in a future automobile SmartSpace, then we need a soft-realtime DSN. A non-realtime system will provide the outputs as fast as possible, with no guarantees and no innate design that supports predictability. Features of an implementation that support this predictability property will be emphasized in the taxonomy.

Depending on the way in which the DSN operates, it is said to be deterministic, quasi-deterministic, or non-deterministic. In a deterministic DSN, it is possible to predict the performance of the DSN accurately. In a quasi-deterministic DSN, although the performance cannot be predicted as accurately as in a deterministic DSN, it is possible to determine worst-case upper bounds within which the DSN can be guaranteed to perform. In a non-deterministic DSN, it is not always possible to guarantee performance and it may be difficult to predict the response time without detailed modeling of various

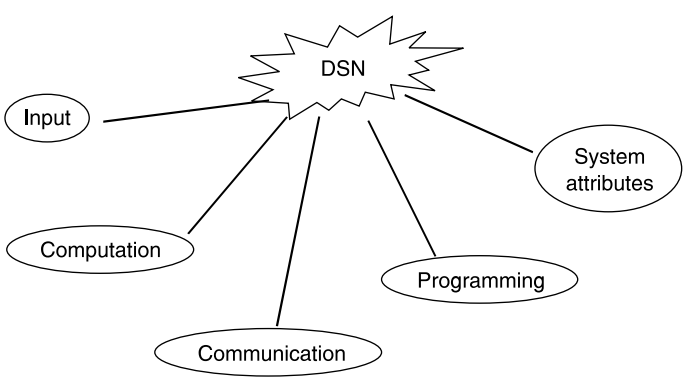


Figure 3.2. Major aspects of a DSN.

© 2005 by Chapman & Hall/CRC

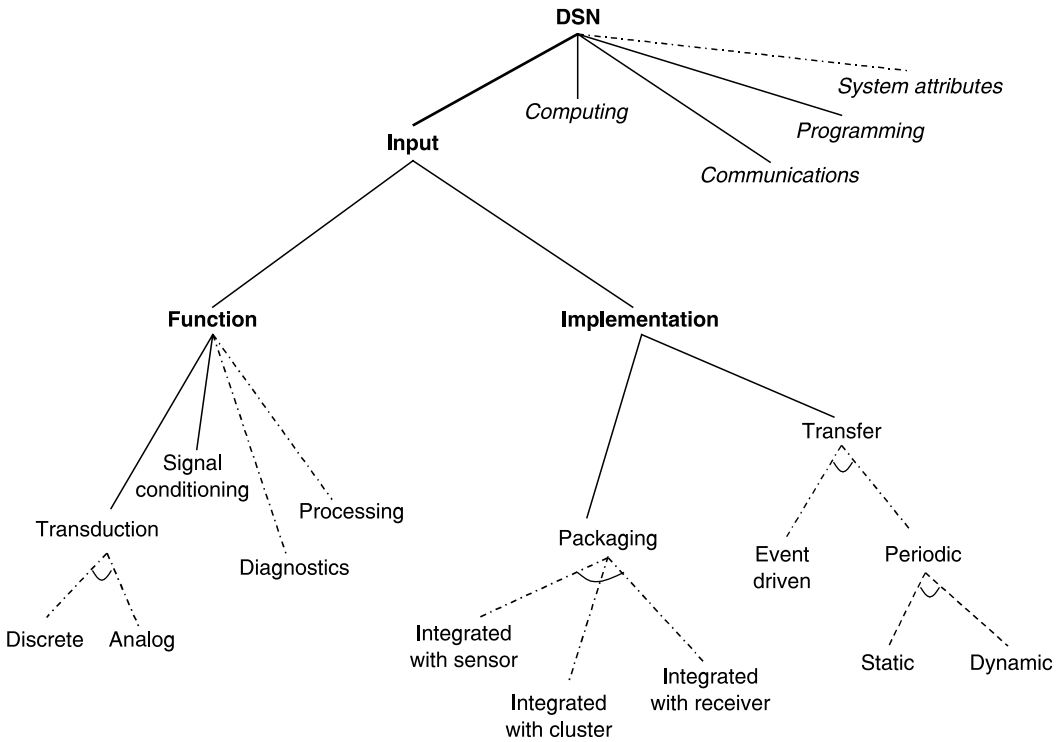


Figure 3.3. DSN input aspect taxonomy.

parameters. Under "normal" operating conditions, the performance of non-deterministic systems can be significantly better than the other systems; however, the behavior under "abnormal" operating conditions is difficult to characterize. The predictability of the various subsystems will be discussed; for example, the communication subsystem or the operating system in sensors can be deterministic, non-deterministic, or quasi-deterministic.

To construct a DSN, one must select at least one of the options in the Input, Computation, Communication and Programming aspects. Zero or more choices may be made among the system attributes depending on a cost/performance balance. In the following taxonomy diagrams, solid lines under a heading indicate choices that are mandatory for a successful system realization; dotted lines indicate optional choices (e.g. see Function in Figure 3.3). A set of dotted lines connected by a solid arc (e.g. see Transduction in Figure 3.3) represents the situation when at least one choice must be made.

3.4.1 Input

We now consider the Input aspect of a DSN as depicted in Figure 3.2. The function of the Input subsystem in a DSN is to capture the input signals from the physical environment and convert the signals to values suitable for processing. As shown in Figure 3.3, there are four primary functions in the Input subsystem: Transduction and Signal Conditioning are mandatory functions, and Diagnostics and Processing are optional functions.

Transduction is either of type analog or discrete. Discrete Input is typically one bit of information (i.e. on/off), whereas analog values may require a substantially larger number of bits and represent continuous values within bounds of representational errors. Data fusion strategies are significantly affected by the type of transduction. For example, several results exist when the sensor values are continuous [9,10]. The theory for discrete type needs further work. Applications tend to be predominantly one or the other, although mixed systems are becoming more prevalent.

© 2005 by Chapman & Hall/CRC

The *Signal Conditioning* function includes activities such as amplifying, digitizing, filtering, forcing, or other signal-processing computations. During input, the result of signal conditioning (irrespective of transduction type) is a digital representation of sensed values. Digital representations are desirable because of their robustness with respect to noise, ability to support error detection and correction schemes, ease of storage and manipulation, and ability to superimpose security mechanisms.

Diagnostics for a sensor refers to methods for determining whether the sensor is functioning properly or has failed. Often, additional circuitry is required for performing self-test at the level of an individual sensor. In certain applications it is feasible to use the values of other sensors, or small history tables, to ascertain deterministically whether or not a sensor has failed.

Processing refers to a list of tasks that may be included at the level of a sensor. Examples of such tasks include historical trending, data logging, alarming functions, and support for configuration and security management.

The *Implementation* of these Input functions significantly affects the overall architecture and performance of the DSN. The transduction function can only be collocated with the sensor, since transduction is the primary purpose of a sensor. The remaining Input functions (i.e. signal conditioning, diagnostics, and processing) may be:

- Located by integrating these functions with the sensor.
- Located in special modules and provide these services to a cluster of sensors.
- Located in devices that use data from the sensors.

These implementation options are shown in Figure 3.3 as *Packaging* options. A particular choice affects the response time (under normal and fault conditions), wiring costs, and the processing requirements of the DSN. By performing the functions locally, we can optimize the implementation with respect to specific devices and avoid contention for communication and DSN resources, thus resulting in faster sampling rates. By locating these functions in a data concentrator, we can reduce the cost of design and maintenance, use more resource-constrained sensor devices, and apply the functions at the level of a cluster. Such a choice tends to increase security risks because of the cost of securing the links between individual links and clusters. The third alternative is to transmit the raw data gathered by sensors directly to receivers in the DSN that requires the data. Typically, this choice tends to increase communication demands and precludes options for early recognition of critical status information.

As an example, consider the diagnostics function. This function can be implemented at the level of a sensor, a cluster, or at the receiver that uses data from the sensor. By locating the diagnostics function at the sensor, we can make local decisions within required time limits. The propagation of erroneous data is prevented. However, we need additional (redundant) circuitry and system resources (memory, timers, etc.) at the sensor level. By performing the diagnostic function at the cluster-level, we reduce the design, implementation, and maintenance costs. It is feasible to use the values of other sensors in the cluster to diagnose a sensor. The resources of sensors can be constrained while the resources at some of the devices (data concentrators) are less constrained and better utilized. On the other hand, if we choose to locate diagnostics at the receivers that use sensed data, then we may require redundant implementations of the function, increase the resource requirements for receivers, and increase the risk of propagating erroneous data in the DSN. The specific choice depends on the application and must be made to balance system-wide cost and performance issues.

The Input *Transfer* function refers to how sensed data is delivered to the DSN and is primarily the responsibility of the communications subsystem. However, from the Input aspect's perspective, the implementation of a transfer method involves the specification of the synchronization method (either periodic or event driven). This choice affects the manner in which the operating system and communication protocols at the level of sensors are designed.

Periodic Input synchronization can either be static or dynamic. Depending on the packaging, such synchronization can be initiated by the DSN by using a master clock, by sensors using local timers, or by data concentrators. Periodic transfer is said to be *static* if the data are gathered deterministically within a fixed time period, called the scan time. The time period for each sensor in the DSN may be different.

© 2005 by Chapman & Hall/CRC

Static-periodic systems have significant, unintentional variations in the scan time. For example, if the strategy is to scan *as-fast-as-possible*, then the scan time is affected if the time to process certain pieces of data is different from others. The scan time also varies when the DSN initiates special processing and transfers in response to certain abnormal events, such as security breaches or multiple sensor faults. Periodic transfer is said to be *dynamic* if successive scan times are not equal. When using dynamic transfer mechanisms, it is important to track both the value and the time at which the data are obtained before synthesizing information.

Event-Driven Input synchronization is fundamentally different from periodic synchronization. It is based on detecting one of the following:

- A change-of-state (COS) of predefined variables.
- Predefined events (as sequences or expressions of COS of predefined variables).

The advantages of an event-driven system over a periodic system are: (1) that it is, on average, more responsive, in the same sense that an Ethernet has a smaller average delay than an equivalent time division multiple access (TDMA) scheme; (2) the amount of communication in a COS-based system can be reduced by not sending repetitive information. However, the disadvantages are: (1) additional measures are necessary to guarantee the delivery of data; (2) methods to detect failures in a sensor–receiver path are required (since it is difficult to distinguish between a failure and a long period of no COS); (3) mechanisms are necessary to prevent an avalanche of data from overwhelming the communications system under exceptional situations. Unlike periodic Input synchronization, event-driven Input synchronization is non-deterministic unless techniques for bounding the performance are used (e.g. priority scheduling).

3.4.2 Computing

The availability of effective communications, coupled with the computational capability of sensors, makes it feasible to host various tasks in sensors. As shown in Figure 3.4, the four primary functions are the Algorithm Processing, Process Diagnostics, Data Management and System Interfaces.

Algorithm Processing concerns tasks that are performed in a sensor node. Specialized algorithms are required to condition signals, encrypt data and process data in the node. Depending on the overall design of the DSN, the nodes may implement components of a distributed algorithm. The operating environment of a node is responsible for ensuring that these algorithms are executed fairly and effectively.

Process Diagnostics are additional computations that are performed at the sensor or cluster levels to augment the Processing function of the Input subsystem. Various techniques for automatically embedding code in the algorithms are being investigated (for diagnostics, monitoring, or distributed services). For example, such embedded code could provide status information and alarm data to operator monitoring stations. Some diagnostic strategies require temporal information in addition to the Input data.

Data Management is another function that is becoming increasingly important for DSNs. Because of the size of contemporary systems, the data gathered by the collection of sensors is immense. Typically, it is not feasible to associate mass storage devices at the level of a sensor, and the amount of memory available in a resource-constrained sensor is limited. Thus, it is necessary to manage the data in a DSN and effectively synthesize information that is useful for decision making. Data management considerations for periodic systems are more critical because of issues of data freshness.

The Computing subsystem must support multiple *System Interfaces* to effectively integrate with other systems. For the interface with the physical environment, it is necessary to interface to proprietary and open sensor interface standards. For example, there are several sensors that interface with Ethernet or SERCOS. To allow users to work with emerging pervasive devices or to incorporate the DSN as an infrastructure for a SmartSpace for Automation [11], the DSN must support open interfaces that are

© 2005 by Chapman & Hall/CRC

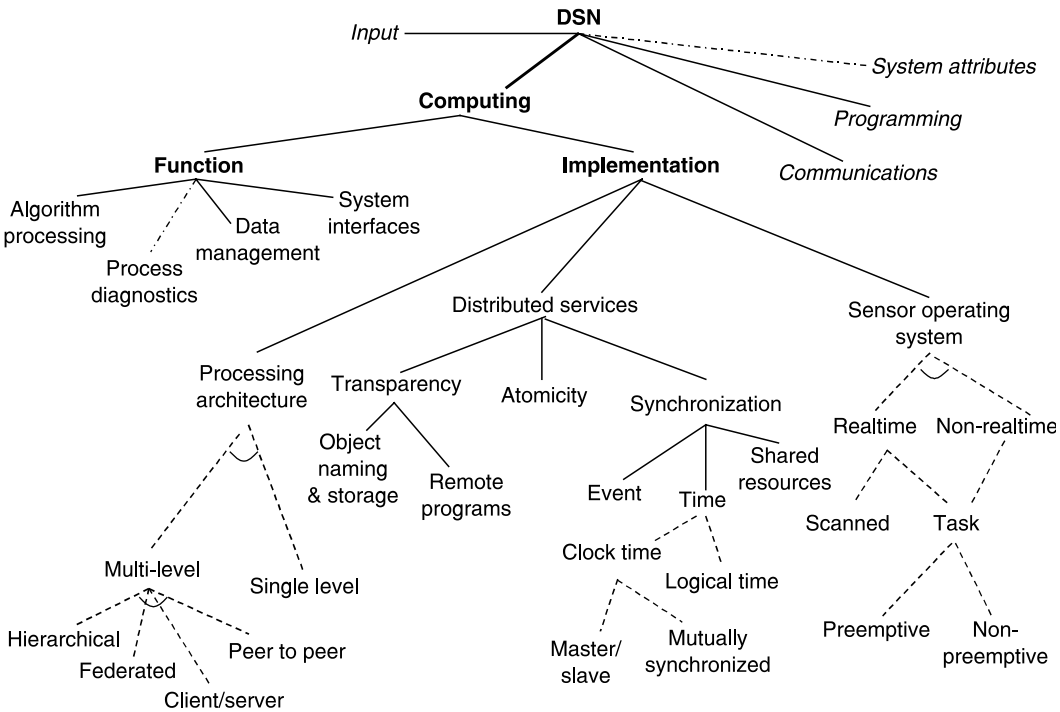


Figure 3.4. DSN computing aspect taxonomy.

based on XML or such other technologies. The implementation of these functions is discussed under the categories of Processing Architecture, Distributed Services and Sensor Operating System.

A DSN is a collection of sensors that are interconnected via some communications media. There are two choices for the *Processing Architecture* of a DSN. In a single-level architecture, all the sensors in the DSN are considered uniformly. Typically, in such an organization, we need to capture and reason about contextual information to manage system evolution properly. Because of the immense scale of DSNs, multi-level architectures are more likely to be successful. There are four common variations of multi-level architecture: (1) *Hierarchical*, in which there are tiers of authority in which sensors in higher tiers are masters of sensors (slaves) in lower tiers of the system. (2) *Federated*, in which certain responsibilities are granted to sensors in a higher tier, but many functions are performed autonomously by sensors in lower tiers. (3) *Client–Server*, in which sensors are delineated into roles so that clients request services or data from the servers. (4) *Peer-to-peer*, in which sensors can be either clients, servers or both. These architectures are not always clearly separable. We expect most systems in the future to be basically federated, with many subsections organized as peer-to-peer or client–server.

Distributed Services facilitate the coding and operation of a DSN and are provided by a distributed operating system that is represented by the collection of operating systems on each sensor. *Transparency* refers to the ability to regard the distributed system as a single computer. Tannenbaum [12] defines several forms of transparency for distributed systems: (1) data or program location, (2) data or process replication, (3) process migration, (4) concurrency and (5) parallelism. For our purposes in a DSN, transparency concerns the *Object Naming and Storage* service, which provides the ability to access system objects without regard to their physical location, and *Remote Program Services*, which provide the ability to create, place, execute, or delete a program without regard to the sensor. Typically, servers are necessary to perform the registration and lookup functions to provide these services.

The *Atomicity* service is used to increase the reliability of the system by insuring that certain operations (called transactions) occur in their entirety, or not at all. Various forms of recovery

© 2005 by Chapman & Hall/CRC

mechanism can be implemented to checkpoint and restore the component state should the atomic operation fail. Typically, atomicity is more important at the level of information-based transactions and less important at the level of periodic data gathering.

The order in which data from various sensors are gathered and the nature of interactions among the multiple sensors depends on the *Synchronization* method. The *Event* service allows a sensor to register an interest in particular events and to be notified when they occur. The *Time* service is used to provide a system-wide notion of time. An important application of system time is in the diagnostic function, where it is used to establish event causality.

Two forms of time are possible: *Clock Time* and *Logical Time*. Providing a system-wide *Clock Time* that is globally known within a specified accuracy to all the DSN nodes can be difficult. Clock Time can represent a standard Universal Coordinated Time (UTC) or it can be a common time local to the system. Two common techniques are: (1) provide a hierarchical *master–slave* system, in which a master sensor in one device is then transmitted to the other slave sensors; (2) use a peer-to-peer distributed mechanism to exchange local times among various sensors. For certain applications it is possible to use global positioning system devices as master clocks to synchronize multiple-nodes with UTC.

Logical Time only provides the relative order of events in the system, not their absolute clock time. For many applications, exact time may not be as important as ensuring that actions occur in the correct sequence, or within certain relative time intervals between events. Many algorithms can be rewritten to use logical time instead of clock time to perform their function. Providing logical clocks in a distributed system may be more cost effective if the applications can be restructured.

The management of *shared resources* across the network is supported through mechanisms that implement mutual exclusion schemes for concurrent access to resources.

All tasks in a sensor execute in an environment provided by the *Sensor Operating System*. This operating system provides services to manage resources, handle interrupts, and schedule tasks for execution. The operating system is said to provide *realtime* services if the length of time required to perform tasks is bounded and predictable. The operating system is said to be *non-realtime* if such services are not supported. Realtime services are supported either by providing a periodic execution model or by providing a realtime scheduler (e.g. rate monotonic scheduling). These schedulers are priority based and can be preemptive (interruptible) or not. Preemptive scheduling can provide the fastest response times, but there is an additional context swap overhead.

Depending on the way in which the scheduler operates, the methods used to code computing, and the interaction with the communication interfaces, then the execution in a sensor can be deterministic, quasi-deterministic, or non-deterministic. One of the main challenges in DSN research is to design efficient deterministic and quasi-deterministic sensor nodes.

3.4.3 Communications

The communication subsystem is the primary infrastructure on which the DSN is constructed, and hence design choices made in this subsystem strongly affect the other capabilities of the DSN. Figure 3.5 presents a taxonomy of this subsystem. The primary functions in this aspect are Data Transport and Bridging.

We distinguish between three types of data and each of these types has different characteristics. *Input data* gathered by sensors is typically limited to a few bytes and needs guaranteed, deterministic message delivery to maintain integrity. Sensors communicate primarily to synchronize and to recover from failures. Thus, *Inter-Sensor* traffic is likely to be sporadic, contain more information (aggregated data), and be more suitable to quasi-deterministic or non-deterministic delivery mechanisms. *System data* refers to all the other data delivery needs that may or may not have hard realtime requirements. For example, data required for system monitoring and status alarms may be critical and realtime, whereas that used by Internet-based supervisory systems may not. Non-realtime system data, such as downloads, can be typically handled in a background mode using a ‘‘best effort’’ protocol.

© 2005 by Chapman & Hall/CRC

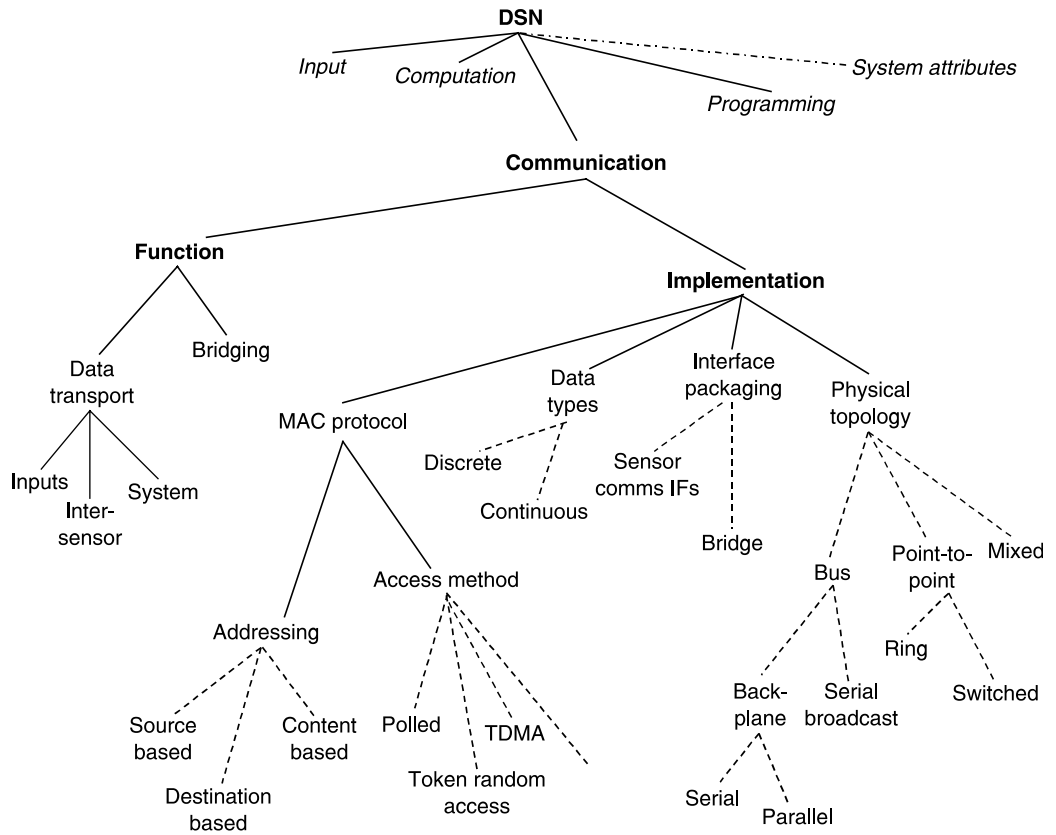


Figure 3.5. DSN communications aspect taxonomy.

The *Bridging* function, which transports data between multiple networks, is important in contemporary distributed systems such as DSNs that are likely to be integrated into existing engineering systems. Bridging refers to tasks performed on interface devices that connect two (or more) networks. The protocol used on the networks may or may not be the same. These intelligent devices provide services such as data filtering, data fusion, alternate routing, and broadcasting and serve to partition the system into logical subsets.

A communication protocol definition, such as in the Open Systems Interface, is designed as layers of services from low-level physical implementation, to media access, through internetworking, up to the application layer. Such layered communication protocols are unlikely to be implemented in resource-constrained sensor nodes. For this taxonomy, we focus only on the media access communication (MAC) layer since it appears to be the layer where most variations occur. Under the *MAC Protocol* implementation attributes we consider two attributes: the addressing scheme and the access mechanism. The method of addressing messages, called the *Addressing Scheme*, can be *source-based*, in which only the producing device's address is used in messages versus using the destination address to route the message. Source-based schemes can be extended to use *content-based* addressing, in which codes are used to identify the type of data within the message. Source- or content-based schemes are typically used on a broadcast bus, a ring, a data server, or when routing schemes can be *a priori* specified. *Destination-based* schemes are used when there is usually one destination or when the routing is constructed dynamically.

The capability to provide deterministic service is strongly affected by the *Access Method* that establishes the rules for sharing the common communication medium. Polled, Token-based and TDMA schemes that use a fixed time slot allocation are deterministic. Token based schemes that allow nodes to

© 2005 by Chapman & Hall/CRC

skip their time slot when they have nothing to transmit, have quasi-deterministic behavior. Random access schemes, such as Ethernet, result in non-deterministic performance, and a priority-bus scheme (e.g. controller area network (CAN)) can be made quasi-deterministic.

The *Data Types* supported on the network is an important design consideration and is related to the type of transduction in the Input aspect of a DSN. If the communication system is optimized for binary or discrete data, then other types of data (e.g. analog) must be transmitted less efficiently. On the other hand, using general protocols precludes the possibility of optimizing special data sets. The choice will be guided by the demands of the application. There may be segregated networks in which the data on each network can be strictly Input data, strictly inter-sensor messages, or strictly system data (e.g. Ethernet); each through a *sensor communications interface* implementing possibly different protocols. Alternatively, the traffic may be mixed on a single network through separate interface devices sharing the media or through a common integrated interface. A bridging function can be packaged as a separate device or integrated with special sensors.

Another influence on the overall architecture is the *Physical Topology* of the communication system. The communication medium (wired or wireless) largely determines the topology in which sensors are connected. Wired systems are *bus based* (single or multiple), a *point-to-point* system, or a combination of the two (e.g. switched busses). The bus-based systems can refer to a local *backplane bus* or to a *serial bus*, as in a local area network (LAN). Typically, bus-based systems are physically able to support message broadcast schemes. Local backplane busses are usually very high speed, relatively short length, and able to perform memory operations at the processor speeds. They can be serial (one data line) or parallel (many data lines). The serial busses used in LANs are typically slower, but they are capable of extending from a few meters to a few kilometers and permitting larger numbers of nodes to be connected. The actual range depends on the protocol, e.g. a token-bus is unlimited in length (except that performance degrades), whereas a CAN bus has a basic limit due to end-to-end propagation time. The point-to-point systems are usually organized as: (1) a ring, in which data are circulated to all devices on the net; (2) switched, to allow messages to be delivered to their specified destinations. Switched topologies can vary from tree-structures to grids to irregular patterns. Several interconnection topologies, such as hierarchical, committee, binary trees, and deBruijn networks, have been considered in the past. With the recent trends in wireless networks, these interconnection topologies are important more for maintaining system cohesion in the presence of changing conditions and less for interconnection.

3.4.4 Programming

This aspect has been largely ignored in the DSN literature. It must cover a range of activities, including designing, developing, debugging, and maintaining programs that perform computing, input, and communication tasks at the sensor levels. Programs must also be developed to support distributed services that are essential for proper functioning of the DSN. In addition, activities such as abnormal state recovery, alarming, and diagnostics must be supported.

Figure 3.6 shows the primary functions of the programming category: support for coding of the *algorithm*, *system testing*, *diagnostics*, *exception handling*, *data management*, *documentation*, and *synchronization*. A key component of each function is the differences that are imposed by having to run in a distributed environment and what services are provided by the programming languages and operating system. For example, the algorithm at a given sensor may require data from another sensor. An issue is whether the data are easily available (transparent services) or whether the programmer must provide code for accessing the remote data explicitly. System testing, diagnostics, and exception handling are complicated by the fact that data are distributed and determination of the true system state is difficult. Documentation includes the program source code and details of system operation. Questions of where programs and documents reside in the distributed system arise, as well as issues in version control and concurrent access. Lastly, the degree of transparency in synchronization that is provided by the languages and environment is a key to simplifying distributed programming.

© 2005 by Chapman & Hall/CRC

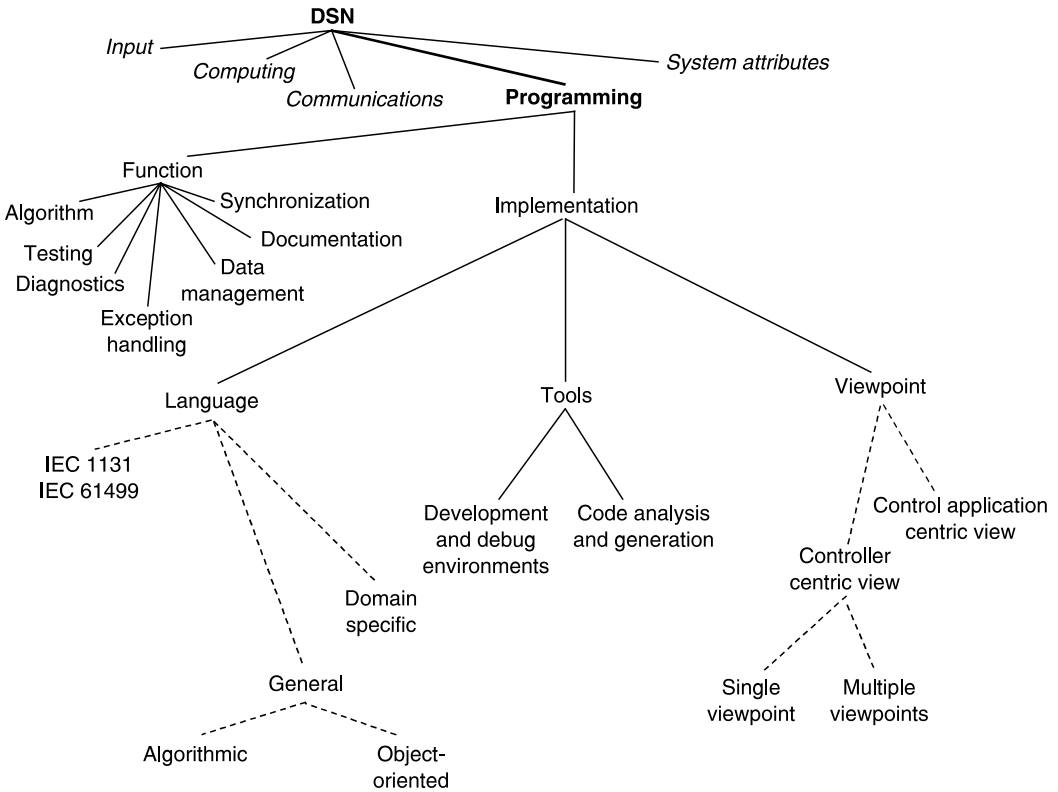


Figure 3.6. DSN programming aspect taxonomy.

The *Language* chosen in a DSN to implement the algorithm affects the services and tools that must provide support (e.g. operating system, compilers, partitioning, performance estimation). The IEC 1131 Programming Standards for digital controllers and the more recent IEC 61499 extension that defines an event-driven execution model are interesting considerations for programming DSNs. Ladder logic is relatively simple to learn, easy to use, and provides a low-level ability to react to process changes. Sequential function charts, Petri nets and finite-state machines (FSMs) are examples of *state-based* languages. An FSM model is intuitively simple, but the size of the model grows rapidly as the number of nodes in the DSN increases. Hierarchical representation methods, such as hierarchical FSMs, have been used to cope with the large size of state-based models. While such hierarchical methods were well suited for hardware design, their use in software design is still an ongoing research issue. Function blocks are designed as a replacement for ladder logic programming in an industrial environment. They provide a graphical, software-IC style language that is simple to use and understand. Function blocks are modular software units, can contain internal states, and represent the inputs and outputs of the function. Libraries of common blocks can be developed to specify node behaviors in a DSN.

Because of immense scale of DSNs, techniques that support the *automatic generation and analysis* of software are important. In an automated code-generation system, the responsibility for managing and maintaining all the interactions between the sensors (by message passing, shared memory, or sharing IO status) is handled automatically. Formal models and theory, such as Petri nets or compiler transformation theory, make the task of software synthesis (and integration) simpler by exploiting the underlying mathematical structure. The user is only responsible for providing a high-level specification of the application needs. In addition, the formal models and theory are also useful for introducing new functionality, such as abnormal state recovery, alarming, and diagnostics.

The *Viewpoint* is another important issue in the Programming aspect. Most of the current programming environments support a *sensor-centric view*. In this view, the needs of the data gathering

© 2005 by Chapman & Hall/CRC

application must be expressed in terms of the capabilities of the sensor that is used in the DSN. When dealing with large applications, managing such programs is a cumbersome activity. In an *application-centric view*, users express data fusion and integration needs by describing relationships among objects in the domain of the application. Application-centric views can be supported with any level of abstraction (i.e. low, medium, or high). However, application-centric views with a low-level abstraction tend to be more akin to a computer-aided drawing for a printed circuit board than a traditional software program. A high level of abstraction is preferable.

A sensor-centric view makes the language more general (i.e. the language and programming environment can be used for different applications). On the other hand, programming a general-purpose application-centric view can be a complex and difficult task.

3.4.5 System Attributes

Several distributed systems support the aspects that are discussed in the preceding sections. DSNs are distinguished by the attributes that are discussed in this section. Teams of researchers are actively investigating all these areas, and we expect that the systems landscape will be significantly changed over the next 2 years.

3.4.6 System Integration

DSNs are unlikely to operate as stand-alone applications. Such systems are likely to be installed as a complementary system in existing engineering infrastructures. Therefore, it is critical for the DSN to integrate with such systems. In particular, the Bridging function of the Communications aspect must provide support for open information and data exchange standards and protocols. Such support is necessary both for active monitoring of the DSN and for passive information gathering from the DSN.

3.4.7 Operating Systems

Architecture issues discussed by Hill *et al.* [5] are relevant for DSNs. TinyOS is an interesting node-level operating system [3,5]. Some of the physical-world integration issues discussed by Estrin *et al.* [14] and Tennenhouse [1] are relevant to DSNs. The reflexive relationship between the computing devices and the application is emphasized and exploited in DSNs by embedding the goal-seeking paradigm (see Emerging trends section 3.3) in the infrastructure. Mechanisms for self-organization in sensor networks [15] are important for a DSN. However, because of the tight integration with the physical world, the performance demands, and the heterogeneous nature of DSN nodes, we require a new, localized, approach to self-organization that preserves determinism, safety, and predictability. The issue of synchronizing time in a sensor network [16] is critical for DSNs and deserves considerable investigation.

3.4.8 Communication Protocols

The large number of nodes in a DSN motivates research into new methods for naming nodes [17] and discovering services [18]. A low-overhead communications mechanism is necessary, and Active Messages [19] is unlikely to provide the low jitter required in certain DSN applications. Ideas of gradients and interests in directed diffusion [20] are likely to be useful for disseminating low-priority information in a DSN during normal operations. SPEED, the new soft-realtime protocol for sensor networks [21], is likely to be adequate for slow, non-demanding, applications.

3.4.9 Security

Distributed security mechanisms are a topic of active research. Security issues of DSNs add to the security issues that are inherited from general distributed systems [22,23], wireless networks [24],

© 2005 by Chapman & Hall/CRC

sensor networks [25,26] and Ethernet-based factory systems [27]. The realtime nature of the DSNs and the rugged operational characteristics that are desired offer new challenges in service discovery, device management and data fusion [2].

3.5 Conclusions

The landscape of architectures of DSNs is vast. The major aspects of a DSN are Input, Computing, Communication, Programming, and System Attributes. The taxonomy proposed here provides a systematic mechanism to traverse this vast landscape. The taxonomy is a useful tool for research planning and system development.

Acknowledgments

This work is supported in part by a University of Akron, College of Engineering Research Startup Grant, 2002–2004, to Dr. Sastry and NSF Grant # IIS-0239914 to Professor Iyengar.

References

[1] Tennenhouse, D., Proactive computing, *Communications of the ACM*, 43(5), 43–50, 2000.
[2] Iyengar, S.S. *et al.*, Foundations of data fusion for automation, *IEEE Instrumentation and Measurement*, 6(4), 35–41, 2003.
[3] Agre, J. *et al.,* A taxonomy for distributed real-time control systems, *Advances in Computers*, 49, 303–352, 1999.
[4] Computer Science Telecommunication Board, *Embedded Everywhere: A Research Agenda for Networked Systems of Embedded Computers*, National Research Council, 2001.
[5] Hill, J. *et al.*, System architecture directions for networked sensors, *ACM Sigplan Notices*, 35, 93–104, 2000.
[6] Iyengar, S.S. *et al.*, Distributed sensor networks for real-time systems with adaptive configuration, *Journal of the Franklin Institute*, 338, 571–582, 2001.
[7] Sathyanarayanan, M., Pervasive computing: vision and challenges, *Pervasive Computing*, (August), 10–17, 2001.
[8] Weiser, M., The computer for the 21st century, *Scientific American,* 265(3), 94–104, 1991.
[9] Iyengar, S.S. *et al.*, A versatile architecture for the distributed sensor integration problem, *IEEE Transactions on Computers*, 43(2), 175–185, 1994.
[10] Marzullo, K., Tolerating failures of continuous-valued sensors, *ACM Transactions of Computing Systems*, 4, 284–304, 1990.
[11] Sastry, S., A SmartSpace for automation, *Assembly Automation*, 24(2), 201–209, 2004.
[12] Tannenbaum, A., *Modern Operating Systems*, Prentice-Hall, NJ, 1992.
[13] Levis, P. and Culler, D., Mate: a tiny virtual machine for sensor networks, in *Architectural Support for Programming Languages and Operating Systems*, 2002.
[14] Estrin, D. *et al.*, Connecting the physical world with pervasive networks, *IEEE Pervasive Computing*, 1(1), 59–69, 2002.
[15] Sohrabi, K. *et al.*, Protocols for self-organization of a wireless sensor network, *IEEE Personal Communications*, 7(5), 16–27, 2000.
[16] Elson, J. and Estrin, D., Time synchronization in wireless sensor networks, in *Proceedings of the 15th International Parallel and Distributed Processing Symposium*, IEEE Computer Society, 2001.
[17] Heidman, J. *et al.*, Building efficient wireless sensor networks with low level naming, in *ACM Symposium on Operating Systems Principles*, 146–159, 2001.
[18] Lim, A.V., Distributed services for information dissemination in self-organizing sensor networks, *Journal of the Franklin Institute*, 338, 707–727, 2001.
[19] Hill, J. *et al.*, Active message communication for tiny network sensors, in *INFOCOM*, 2001.

© 2005 by Chapman & Hall/CRC

[20] Intanagonwiwat, C. *et al.*, Directed diffusion: a scalable and robust communication paradigm for sensor networks, in *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, 56–67, 2000.
[21] He, T. *et al.*, SPEED: a stateless protocol for real-time communication in sensor networks, Technical Report, University of Virginia, 2002.
[22] Caelli, W.J., Security in open distributed systems, *Information Management and Computer Society*, 2(1), 18–24, 1994.
[23] Horrell, M. and Limited, Z., The security of distributed systems — an overview, *Information Security Technical Report*, 1(2), 10–16, 1996.
[24] National Institute of Science and Technology, *DRAFT: Wireless Network Security*, NIST, Special Publication 800-48, Computer Security Division, 2002.
[25] Perrig, A. *et al.*, SPINS: security protocols for sensor networks, in *MOBICOM*, 189–199, 2001.
[26] Wood, A.D. and Stankovic, J.A., Denial of service in sensor networks, *IEEE Computer*, 54–62, 2002.
[27] Siemens, *Information Security for Industrial Communications*, White Paper, 1999.

© 2005 by Chapman & Hall/CRC

4

Contrast with Traditional Systems

R.R. Brooks

4.1 Problem Statement

Sensor networks are a fundamentally new type of system. Computing systems up to now have been primarily user-centric. Desktop devices interact directly with a human operator following their instructions. Networked systems served primarily as communications devices between human users. These communications were often augmented by the existence of databases for transaction processing. Information was retained and processed for later use.

Sensor networks now provide distributed systems that interact primarily with their environment. The information extracted is processed and retained in databases as before, but the human is removed from many parts of the processing loop. Devices are designed to act more autonomously than was previously the case. They are also designed to work as a team.

Embedded systems are not new, but sensor networks greatly extend the capabilities of these systems. Up to now, embedded systems were resource-constrained devices providing limited intelligence in a strictly defined workspace. Sensor network research is aiming towards developing self-configuring systems working in either unknown or inherently chaotic environments.

Network communications needs to be handled in a new manner. Data are important because of their contents and not because of the machine from where it originates. Wireless transmission is expensive, making it attractive to process data close to the source. The problems of multi-path fading, interference, and limited node lifetimes combine to make data paths, even under good conditions, transient and unreliable. The implications this has for network design should not be underestimated.

Similarly, signal-processing technologies need to be aware of transmission delays and the volume of data needed. The resources consumed by an implementation, and the latencies incurred, are an integral part of any sensing solution. In short, all levels of the networking protocols need to be considered as a single gestalt. Beyond this, system design requires new power-aware computing hardware and operating systems.

Embedded sensors are now part of large, loosely coupled networks with mobile code and data. Data supply and demand are stochastic and unpredictable. Processing occurs concurrently on multiple

© 2005 by Chapman & Hall/CRC

processors. Applications are in hostile environments, with noise-corrupted communication and failure-prone components. Under these conditions, efficient operation cannot rely on static plans. Van Creveld [1] has defined five characteristics that hierarchical systems need to adapt to this type of environment [2]:

- Decision thresholds fixed far down in the hierarchy.
- Self-contained units exist at a low level.
- Information circulates from the bottom up and the top down.
- Commanders actively seek data to supplement routine reports.
- Informal communications are necessary.

Organizations based on this approach have been successful in market economies, war, and law enforcement [3].

Data requests from a declarative programming language *pull* data by making requests. Sensors *push* information by providing sensor data. Requests and data follow paths of least resistance through intermediate nodes. Mobile data and code allow signal processing and automatic target recognition to be done during data transmission. Processing includes decomposition, compression, and fusion of sensor data. This is done robustly and efficiently by giving each node the ability to make limited local optimizations based on locally available information.

The sensor network consists of nodes integrating these abilities. Requests for information form flexible *ad hoc* virtual enterprises of nodes, allowing the network to adapt to and compensate for failures and congestion. Complex adaptive behavior for the whole network emerges from straightforward choices made by individual nodes.

The final system will be a virtual enterprise. Groups of components form flexible *ad hoc* confederations to deliver data in response to changing needs and resources. This is applicable to any sensing modality and can be implemented on any testbed containing networked processors, sensors, and embedded processors.

Factors needing to be considered for system optimization include:

- Data format for transmission.
- Paths taken through the network for requests and data.
- Points where fusion should occur during transmission.
- Processing to be performed by mobile code during data transit.

The final issue to consider is adaptation to system state. Information can be exchanged in a number of formats. It is reasonable to compress data for transmission over slow channels. Transmission over noisy channels requires redundant data. As noise (lack of noise) is detected in a channel, error checking increases (decreases). The meta-protocol starts with pessimistic assumptions of channel quality and modifies the protocol dynamically. Modifications are based on information from normal operations. Extra traffic for monitoring status is to be avoided.

Sensor networks are fundamentally different from their predecessors. Some of these differences are a matter of degree:

- Number of nodes required
- Power constraint severity
- Proximity to hostile environment
- Timeliness requirements

In the final analysis, the most important differences are the fundamental differences in the way information technology is used. Computer networks are no longer sensitive devices being coddled in air-conditioned clean rooms. They are working in hostile conditions. Sensor networks respond to the needs of human users. How they respond, and their internal configurations, will be decided independent of human intervention.

© 2005 by Chapman & Hall/CRC

Acknowledgments and Disclaimer

This research is sponsored by the Defense Advance Research Projects Agency (DARPA), and administered by the Army Research Office under Emergent Surveillance Plexus MURI Award No. DAAD19-01-1-0504. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author and do not necessarily reflect the views of the sponsoring agencies.

References

[1] van Crefeld, M.L., *Command in War,* Harvard University Press, Cambridge, MA, 1985.
[2] Czerwinski, T., *Coping with the bounds: speculations on nonlinearity in military affairs*, National Defense University, Washington, DC.
[3] Cebrowski, A.K. and Garsta, J.J., Network-centric warfare: its origin and future, *Proceedings of the Naval Institute*, January 1998. http://www.usni.org/Proceedings/Articles98/PROcebrowski.htm.

© 2005 by Chapman & Hall/CRC

II

Distributed Sensing and Signal Processing

5. **Digital Signal Processing Backgrounds** *Yu Hen Hu* *53*
Introduction • Discrete-Time System Theory • Frequency Representation and the DFT • Digital Filters • Sampling, Decimation, Interpolation • Conclusion • Appendix 5.1 • Appendix 5.2 • Appendix 5.3 • Appendix 5.4

6. **Image-Processing Background** *Lynne Grewe and Ben Shahshahani* *71*
Introduction • Motivation • Image Creation • Image Domains: Spatial, Frequency and Wavelet • Point-Based Operations • Area-Based Operations • Noise Removal • Feature Extraction • Registration, Calibration, Fusion Issues • Compression and Transmission: Impacts on a Distributed Sensor Network • More Imaging in Sensor Network Applications

7. **Object Detection and Classification** *Akbar M. Sayeed* *97*
Introduction • A Signal Model for Sensor Measurements • Object Detection • Object Classification • Conclusions

8. **Parameter Estimation** *David S. Friedlander* *115*
Introduction • Self-Organization of the Network • Velocity and Position Estimation • Moving Target Resolution • Target Classification Using Semantic Information Fusion • Stationary Targets • Peaks for Different Sensor Types • Acknowledgments

9. **Target Tracking with Self-Organizing Distributed Sensors** *R.R. Brooks, C. Griffin, David S. Friedlander, and J.D. Koch* *135*
Introduction • Computation Environment • Inter-Cluster Tracking Framework • Local Parameter Estimation • Track Maintenance Alternatives • Tracking Examples • The CA Model • CA results

© 2005 by Chapman & Hall/CRC

• Collaborative Tracking Network • Dependability Analysis
• Resource Parsimony • Multiple Target Tracking
• Conclusion • Acknowledgments

10. **Collaborative Signal and Information Processing: An Information-Directed Approach** *Feng Zhao, Jie Liu, Juan Liu, Leonidas Guibas, and James Reich* 185
Sensor Network Applications, Constraints, and Challenges • Tracking as a Canonical Problem for CSIP • Information-Driven Sensor Query: A CSIP Approach to Target Tracking • Combinatorial Tracking Problems • Discussion • Conclusion • Acknowledgments

11. **Environmental Effects** *David C. Swanson* 201
Introduction • Sensor Statistical Confidence Metrics • Atmospheric Dynamics • Propagation of Sound Waves

12. **Detecting and Counteracting Atmospheric Effects** *Lynne L. Grewe* 213
Motivation: The Problem • Sensor-Specific Issues • Physics-Based Solutions • Heuristics and Nonphysics-Based Solutions • Conclusions

13. **Signal Processing and Propagation for Aeroacoustic Sensor Networks** *Richard J. Kozick, Brian M. Sadler, and D. Keith Wilson* 225
Introduction • Models for Source Signals and Propagation • Signal Processing • Concluding Remarks • Acknowledgments

14. **Distributed Multi-Target Detection in Sensor Networks** *Xiaoling Wang, Hairong Qi, and Steve Beck* 271
Introduction • The BSS Problem • Source Number Estimation • Distributed Source Number Estimation • Performance Evaluation • Conclusions

This section discusses signal processing and sensor data interpretation issues of distributed sensor networks (DSNs). The chapters presented are tutorial in nature. Some chapters are overviews and surveys of important issues and others delve into recent advances in the field. Every effort has been made to make the material accessible to technically literate readers.

Hu provides a survey and review of digital signal processing. His chapter describes what signals are, and the most widely used transformations for time series data. The discussion contains a number of practical examples.

Grewe and Shahshahani explain the basics of image processing. Where signal processing deals primarily with one-dimensional time series, image processing concentrates on two-dimensional images. These images are mainly in the visible wavelengths. They describe how imaging devices work, and provide transformations that aid in extracting information from the images. Of particular interest is the discussion of image calibration, registration, and data assoication.

Sayeed discusses methods for object detection and classification. Detection occurs when a signal of interest is located in a set of readings. Classification refers to differentiating between signals emanating from different classes of targets. For example, a tripwire sensor can usually detect the presence of a vehicle but is unable to differentiate between vehicle types. On the other hand, an acoustic sensor may be able to differentiate between tracked and wheeled vehicles. Sayeed develops a model describing sensor readings. He then provides statistical techniques for differentiating between signals belonging to classes of interest and background noise. In doing so, he concentrates on issues related to interpreting multiple sensor inputs.

© 2005 by Chapman & Hall/CRC

Friedlander describes techniques that use distributed sensor inputs to estimate parameters that describe targets. He provides a novel technique for estimating target heading and velocity. Results from a field test are given showing that this approach is very robust. He also explains how sensor networks can be used to count the number of targets present in the sensor field. This problem is deceptively difficult, and he derives limitations on the networks ability to perform this task.

Brooks *et al.* describes a distributed tracking approach that uses the parameter estimation technique described in Friedlander's chapter. A clump head is chosen locally to estimate the heading and velocity of a target based on the local information. This data is propagated through the network to those nodes likely to see the target in the future. The chapter describes several techniques that have been tested for associating target detections to tracks.

An alternative tracking approach is discussed by Zhao *et al.* Sensors are considered as providers of information for the system. Nodes evaluate their current beliefs and determine what information is needed to disambiguate their beliefs. For target counting, this approach leads to electing leaders for equivalence classes of nodes that detect the same target.

Swanson discusses the problems faced by sensor nodes in the real world. Environmental defects on sensor inputs are very difficult to account for. The effect of wind and weather on acoustic sensors make it very difficult to design robust algorithms that work dependably. The problem of calibrating seismic sensors to accommodate differences in soil and bedrock have made them very difficult to use in rapidly deployed systems. Swanson's discussion of the atmospheric issues that make the detection of chemical and biological agents challenging will be of interest to many readers.

Grewe discusses how to counteract some of these environmental effects in imaging systems. She explains how atmospheric effects degrade the performance of imaging sensors. Results are given showing how the presence of fog in images can be detected and mitigated.

Kozick *et al.* discuss the use of narrow and wide band data sources in sensor networks. They provide models of signals and their propagation. These models are used to derive methods for estimating the angle of arrival of a target and localizing targets.

Wang *et al.* discusses the problem of detecting multiple targets in a distributed network. This chapter initially discusses methods for detecting the number of signal sources in readings. They then delve into the more complicated issue of combining these estimates into a hierarchical network. A Bayesian approach is proposed and field-tested.

This section provides essential background on signal processing issues. Readers will find that the discussion progression from tutorial information to in-depth discussion of research issues. A number of topics, such a signal detection, classification, and target tracking are viewed from many different perspectives. These issues are one of the core aspects of sensor networks. Other aspects arise from these approaches being embedded in an unreliable distributed system.

© 2005 by Chapman & Hall/CRC

5

Digital Signal Processing Backgrounds

Yu Hen Hu

5.1 Introduction

The purpose of this chapter is to review the fundamentals of deterministic digital signal processing techniques, with specific attention to wireless distributed sensor network applications. Signal processing concerns the acquisition, filtering, transformation, estimation, detection, compression, and recognition of signals represented in multiple media, multiple modalities, including sound, speech, image, video, and others. In *digital* signal processing, a natural or synthetic signal is first sampled and quantized using an analog-to-digital (A/D) converter. The result is a stream of finite-precision numbers that can be processed using a digital computer. The results may then be converted back to a continuous-time analog form using a digital-to-analog (D/A) converter.

Examples of natural signals in sensor network applications include temperature, humidity, wind speed, density of chemical agents, gas, solvent, sound, voice, image and video of targets, gesture, facial expressions, and many others. Essentially, every type of sensor will produce a stream of signals in the form of time-varying electrical voltage or current waveforms. Many sensor nodes also have built-in A/D converters sampling the signal at a prespecified sampling rate.

After sampling, the sampled signals need to be preprocessed before applying additional collaborative signal-processing algorithms. The preprocessing step may involve digital filtering, sampling rate conversion, discrete Fourier transform (DFT) and other deterministic digital signal processing operations.

The preprocessed digital signal may then be subject to further on-board signal processing so that the amount of data that needs to be transmitted through a wireless channel can be reduced. This will not only reduce the network congestion and hence improve communication efficiency, but also conserves the rate of energy consumption at individual sensor nodes. Deterministic signal processing tasks performed at this stage may involve discrete cosine transform, discrete wavelet transform (for data compression), and digital filtering.

© 2005 by Chapman & Hall/CRC

In the rest of this chapter, the following topics will be introduced. First, the basic discrete-time system theory will be reviewed including basic definitions and properties. Next, the frequency domain representation of discrete time signals, the DFT, and fast Fourier transform (FFT) implementation will be reviewed. The topic of digital filters will then be discussed. This is followed by the introduction of a sampling theorem, decimation and interpolation techniques.

5.2 Discrete-Time System Theory

5.2.1 Discrete Time Signal

A discrete time signal, denoted $x[n]$, is a sequence of real-valued or complex-valued finite-precision numbers that are sampled from a continuous-time signal $x(t)$ at a regular interval T such that $x[n] = x(nT)$. For convenience, the time indices n may be shifted during processing so that the sequence may contain negative indices. The indices of a sequence n may range from $-\infty$ to $+\infty$. Hence, mathematically, a sequence can have infinite length. In reality, only sequences with finite length can be processed.

Some examples of basic sequences are listed in Table 5.1, and several properties of a sequence are introduced in Table 5.2 without extensive discussion. For more details, see Mitra [1] and Oppenheim and Schafer [2].

5.2.2 Discrete-Time System

A *system* maps sequences to sequences. A single-input–single-output (SISO) discrete-time system is a *mapping* from an input sequence $x[n]$, to an output sequence, also known as a *response*, $y[n]$. Therefore,

Table 5.1. Examples of discrete-time sequences

Type	Definition
Unit sample (impulse) sequence	$\delta[n] = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0 \end{cases}$
Unit step sequence	$u[n] = \begin{cases} 1, & n \geq 0 \\ 0, & n < 0 \end{cases}$
Sinusoidal sequence	$x[n] = A\cos(\omega_0 n + \phi)$
Exponential sequence	$x[n] = A\alpha^n$

Table 5.2. Properties of discrete-time sequences

Property	Formula (for all n)
Conjugate-symmetric, even	$x[n] = x^*[-n]$
Conjugate-antisymmetric, odd	$x[n] = -x^*[-n]$
Periodic with period N	$x[n] = x[n + kN]$; k: integer
Energy of a sequence $x[n]$	$E = \sum_{n=-\infty}^{\infty} \lvert x[n]\rvert^2$
Bounded sequence	$\lvert x[n]\rvert \leq B_x < \infty$
Square-summable sequence	$\sum_{n=-\infty}^{\infty} \lvert x[n]\rvert^2 < \infty$

© 2005 by Chapman & Hall/CRC

Table 5.3. Properties of discrete-time systems

Type	Definition
Linear	Let a and b be two constants. The system is a linear system if and only if $x[n] = ax_1[n] + bx_2[n]$ implies $y[n] = ay_1[n] + by_2[n]$
Shift (time) invariant	Let n_0 be a fixed integer. The system is shift invariant if and only if $x[n] = x_1[n - n_0]$ implies $y[n] = y_1[n - n_0]$
Causal	Let n_0 be a fixed integer. The system is causal if and only if $x_1[n] = x_2[n]$ for $n < n_0$ implies $y_1[n] = y_2[n]$ for $n < n_0$
Stability	Let A and B be two appropriately chosen constants. A system is bounded-input, bounded-output (BIBO) stable if and only if $x[n] < A\ \forall n$ implies $y[n] < B\ \forall n$

it is possible that a particular output value $y[n_1]$ depends not only on corresponding input value $x[n_1]$, but also on other input values $x[n]$, $n \neq n_1$. Some examples of discrete-time systems are:

$$y[n] = \sum_{k=0}^{\infty} x[n-k] = \sum_{m=-\infty}^{n} x[m]$$

$$y[n] = 3x[n-1] + 2x[n+1]$$

Some important properties of discrete-time systems are summarized in Table 5.3. For convenience, assume that $y_1[n]$ and $y_2[n]$ are the responses corresponding to the input sequences $x_1[n]$ and $x_2[n]$ respectively.

5.2.3 Impulse Response Characterization of a Linear and Shift (Time) Invariant Discrete-Time System

If a system is both linear and shift (time) invariant, it is called an LTI system. An LTI system can be uniquely characterized by its *impulse response* $h[n]$, defined as the response of the system when the input is an impulse sequence. In other words, given the impulse response $h[n]$ and an input sequence $x[n]$, the corresponding response of an LTI system, denoted by $y[n]$, can be found through the following convolution operation:

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k] = \sum_{k=-\infty}^{\infty} h[k]x[n-k]$$

If the impulse response sequence has finite length, i.e.

$h[n] = 0$ for $n < 0$, and $n > N$

then it is called a *finite* impulse response (FIR). Otherwise, it is an *infinite* impulse response (IIR).

5.3 Frequency Representation and the DFT

5.3.1 The *z*-Transform

The z-transform is a complex polynomial representation of a discrete time sequence. Given x$[n]$, its z-transform is defined as

$$X(z) = \sum_{n=-\infty}^{\infty} x[n]z^{-n} \tag{5.1}$$

© 2005 by Chapman & Hall/CRC

Table 5.4. The z-transform pairs of some common sequences

Sequence	z-transform	ROC
$\delta[n]$	1	Entire z-plane
$u[n]$	$1/(1-z^{-1})$	$\|z\|>1$
$a^n u[n]$	$1/(1-az^{-1})$	$\|z\|>\|a\|$
$r^n \cos(\omega_0 n)u[n]$	$\dfrac{1-(r\cos\omega_0)z^{-1}}{1-(2r\cos\omega_0)z^{-1}+r^2z^{-2}}$	$\|z\|>\|r\|$
$r^n \sin(\omega_0 n)u[n]$	$\dfrac{1-(r\sin\omega_0)z^{-1}}{1-(2r\cos\omega_0)z^{-1}+r^2z^{-2}}$	$\|z\|>\|r\|$

where z is a complex variable over the complex z-plane. The region of convergence (ROC) of $X(z)$ is the region in the z-plane where $|X(z)|$ is finite. The z-transform pairs of some popular sequences are listed in the Table 5.4

Among the many interesting properties of the z-transform, perhaps the most useful is the convolution property that states if $y[n]$ is the results of the convolution of two sequences $x[n]$ and $h[n]$, and $Y(z)$, $X(z)$, $H(z)$, respectively are their corresponding z-transforms, then

$$Y(z) = X(z)H(z) \tag{5.2}$$

If the input sequence is a unit-sample (impulse) sequence, then $X(z)=1$ according to Table 5.4. Hence, $Y(z)=H(z)$. Since $H(z)$ is the z-transform of the impulse response $h[n]$, it is called the *system function* or the *transfer function* of the underlying system.

The z-transform representation is useful in practical signal-processing applications in several ways. Firstly, for finite length sequence, its z-transform is a polynomial with a finite number of terms. As shown in Equation (5.2), the convolution of the two sequences can be easily obtained by multiplying the corresponding polynomials. Secondly, for a broad class of LTI systems, their transfer functions can be represented well with a quotient of two polynomials, such as those shown in Table 5.4. When an LTI system is represented in such an expression, it is possible to solve its time response analytically, and to analyze its behavior in great detail. Let us assume $A(z)$ and $B(z)$ are two finite polynomials such that

$$H(z) = \frac{A(z)}{B(z)} = K\frac{\prod_{k=1}^{P}(z-z_k)}{\prod_{l=1}^{Q}(z-p_l)} \tag{5.3}$$

where $\{z_k;\ 1 \le k \le P\}$ are called *zeros* and $\{p_l; 1 \le l \le Q\}$ are called *poles* of the transfer function $H(z)$.

An LTI system is *stable* if all the poles of its transfer function locate within the unit circle $\{z;\ |z|=1\}$ in the z-plane.

5.3.2 Discrete-Time Fourier Transform

The discrete-time Fourier transform (DTFT) pair of a sequence $x[n]$, denoted by $X(e^{j\omega})$, is defined as

$$X\left(e^{j\omega}\right) = \sum_{n=-\infty}^{\infty} x[n]\, e^{-j\omega n}$$

$$x[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} X\left(e^{j\omega}\right) e^{j\omega n}\, d\omega \tag{5.4}$$

© 2005 by Chapman & Hall/CRC

Note that $X(e^{j\omega})$ is a periodic function of ω with period equals to 2π. The DTFT and the z-transform are related such that

$$X\left(e^{j\omega}\right) = X(z)\big|_{z=e^{j\omega}}$$

5.3.3 Frequency Response

The DTFT of the impulse response $h[n]$ of an LTI system is called the *frequency response* of that system, and is defined as

$$H\left(e^{j\omega}\right) = \text{DTFT}\left\{h[n]\right\} = H(z)\big|_{z=e^{j\omega}} = \left|H\left(e^{j\omega}\right)\right| e^{j\theta(\omega)}$$

where $|H(e^{j\omega})|$ is called the *magnitude response* and $\theta(\omega) = \arg\{H(e^{j\omega})\}$ is called the *phase response*. If $\{h[n]\}$ is a real-valued impulse response sequence, then its magnitude response is an even function of ω and its phase response is an odd function of ω. The derivative of the phase response with respect to frequency ω is called the *group delay*. If the group delay is a constant at almost all ω, then the system is said to have *linear phase*. If an LTI system has unity magnitude response and is linear phase, then its output sequence will be a delayed version of its input without distortion.

5.3.4 The DFT

For real-world applications, only finite-length sequences are involved. In these cases, the DFT is often used in lieu of the DTFT. Given a finite-length sequence $\{x[n]; 0 \le n \le N-1\}$, the DFT and inverse DFT (IDFT) are defined thus:

$$\begin{aligned}
&\text{DFT:} \quad X[k] = \sum_{n=0}^{N-1} x[n] \exp\left(-\frac{j2\pi kn}{N}\right) = \sum_{n=0}^{N-1} x[n] W_N^{kn} \quad 0 \le k \le N-1 \\
&\text{IDFT:} \quad x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k] \exp\left(\frac{j2\pi kn}{N}\right) = \frac{1}{N}\sum_{k=0}^{N-1} X[k] W_N^{-kn} \quad 0 \le n \le N-1
\end{aligned} \tag{5.5}$$

where $W_N = e^{-j2\pi/N}$.

Note that $\{X[k]\}$ is a periodic sequence in that $X[k+mN] = X[k]$ for any integer m. Similarly, the $x[n]$ sequence obtained in equation (5.5) is also periodic, in that $x[n+mN] = x[n]$. Some practically useful properties of the DFT are listed in Table 5.5, where the *circular shift operation* is defined as,

$$x[\langle n - n_0 \rangle_N] = \begin{cases} x[n-n_0] & n_0 \le n \le N-1 \\ x[n-n_0+N] & 0 \le n < n_0 \end{cases}. \tag{5.6}$$

The convolution property is very important, in that the response of an LTI system can be conveniently computed using the DFT if both the input sequence $x[n]$ and the impulse response sequence $h[n]$ are finite-length sequences. This can be accomplished using the following algorithm:

Algorithm. Compute the output of an FIR LTI system.
Given: $\{x[n]; 0 \le n \le M-1\}$ and $\{h[n]; 0 \le n \le L-1\}$:

1. Let $N = M+L-1$. Pad zeros to both $x[n]$ and $h[n]$ so that they both have length N.

© 2005 by Chapman & Hall/CRC

Table 5.5. Properties of DFT

Property	Length-N sequence	N-point DFT
	$x[n]$, $y[n]$	$X[k]$, $Y[k]$
Linearity	$ax[n] + by[n]$	$aX[K] + bY[k]$
Circular shift	$x[\langle n - n_0 \rangle_N]$	$W_N^{kn_0} X[k]$
Modulation	$W_N^{-k_0 n} x[n]$	$X[\langle k - k_0 \rangle_N]$
Convolution	$\sum_{m=0}^{N-1} x[m] y[\langle n - m \rangle_N]$	$X[k]Y[k]$
Multiplication	$x[n]y[n]$	$\frac{1}{N}\sum_{m=0}^{N-1} X[m]Y[\langle k - m \rangle_N]$

2. Compute the respective DFTs of these two zero-padded sequences, $X[k]$ and $H[k]$.
3. Compute $Y[k] = X[k]H[k]$ for $0 \le k \le N-1$.
4. Compute $y[n] = \text{IDFT}\{Y[k]\}$.

There are some useful symmetry properties of the DFT that can be explored in practical applications. We focus on the case when $x[n]$ is a real-valued sequence. In this case, the following symmetric relations hold:

$$X[k] = X^*[N-1-k]$$

Therefore, one may deduce

$$\text{Re } X[k] = \text{Re } X[N-1-k]$$
$$\text{Im } X[k] = -\text{Im } X[N-1-k]$$
$$|X[k]| = |X[N-1-k]|$$
$$\arg X[k] = \arg X[N-1-k]$$

where

$$\arg X[k] = \tan^{-1}\frac{\text{Im } X[k]}{\text{Re } X[k]} \in \left[-\pi \quad \pi\right]$$

5.3.5 The FFT

The FFT is a computation algorithm that computes the DFT efficiently. The detailed derivation of the FFT is beyond the scope of this chapter. Readers who are interested in knowing more details about the FFT are referred to several excellent textbooks, e.g. [1–3].

5.4 Digital Filters

Digital filters are LTI systems designed to modify the frequency content of the input digital signal. These systems can be SISO systems or multiple-inputs–multiple-outputs (MIMO) systems.

5.4.1 Frequency Response of Digital Filters

Depending on the application, the frequency response of a digital filter can be characterized as all pass, band-pass, band-stop, high-pass, and low-pass. They describe which frequency band of the

© 2005 by Chapman & Hall/CRC

input sequence is allowed to pass through the filter while the remaining input signals are filtered out. All pass filters are often implemented as a MIMO system such that the input sequence is decomposed into complementary frequency bands. These components can be combined to perfectly reconstruct the original signal with a fixed delay. Hence, the overall system passes all the frequency bands and, therefore, the term *all pass*. With a MIMO system, the digital filter has a *filter bank* structure that can be exploited to implement various linear transformations, including DFT and discrete wavelet transform (DWT). Digital filter banks have found wide acceptance for applications such as data compression, multi-resolution signal processing, and orthogonal frequency division multiplexing. Low-pass filters are perhaps the most commonly encountered digital filter. They have found applications in removing high-frequency noise, extracting the low-frequency trend, and preventing alias before decimation of a digital sequence. High-pass filters are used for exposing the high-frequency content of a potential signal. They can be used for event detection. A band-stop filter will filter out unwanted interference from a frequency band that does not significantly overlap with the desired signal. For example, in a special band-stop filter, known as the notch-filter, it is possible to reject 60 Hz power-line noise without affecting the broadband signal. Band-pass digital filters are designed to pass a narrow-band signal while rejecting broadband background noise. Table 5.6 illustrates the magnitudes of that frequency responses of four types of digital filter. The corresponding transfer functions are also listed with $a = 0.8$ and $b = 0.5$. The constants are to ensure that the maximum magnitudes of the frequency responses are equal to unity. The Matlab® program that generates these plots is given in Appendix 5.1.

5.4.2 Structures of Digital Filters

Based on whether the impulse response sequence is of finite length, digital filter structures can be categorized into FIR filters and IIR filters. An FIR filter has several desirable characteristics:

1. It can be designed to have exact an linear phase.
2. It can easily be implemented to ensure the BIBO stability of the system.
3. Its structure is less sensitive to quantization noise than an IIR filter.

In addition, numerous computer-aided design tools are available to design an arbitrarily specified FIR filter with relative ease.

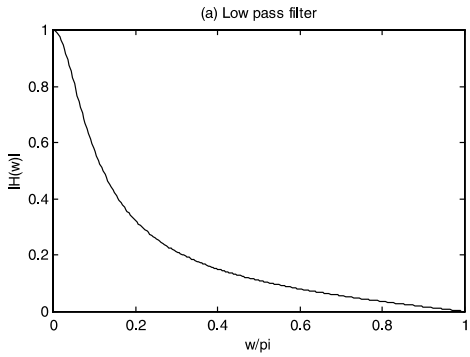
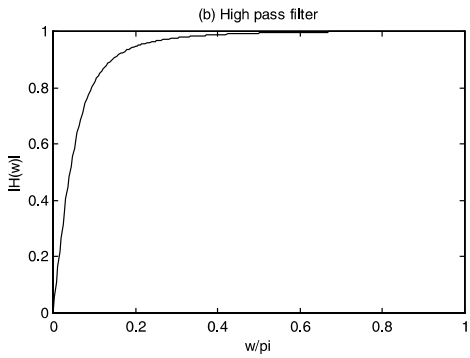
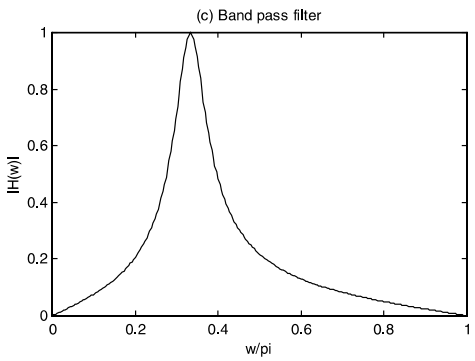
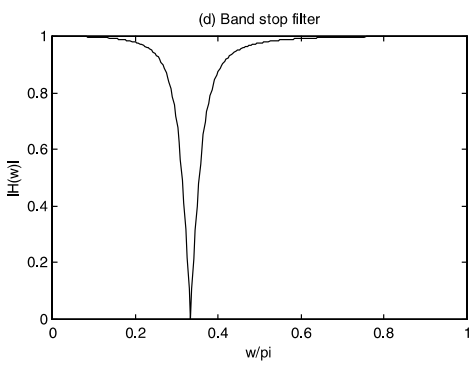
Compared with the FIR filter, an IIR filter often requires fewer computation operations per input sample, and hence would potentially consumes less power on computing. However, the stability of an IIR filter is prone to accumulation of quantization noise, and linear phase usually cannot be guaranteed.

5.4.3 Example: Baseline Wander Removal

Consider a digital signal sequence $\{x[n];\ 0 \le n \le 511\}$ as plotted in Figure 5.1(a) by the dotted line. A low-pass FIR filter is designed to have the impulse response shown in Figure 5.1(b). This FIR filter is designed to use the Hamming window with $2L + 1$ non-zero impulse response components. In this example, $L = 20$, and $2L + 1 = 41$. It has a default normalized cut-off frequency $10/m$, where m is the length of the sequence (512 here). At the cut-off frequency, the magnitude response of the FIR filter is half of that at the zero frequency. The magnitude frequency response of the FIR filter, represented in decibel (dB) format, is shown in Figure 5.2(a). In general, it contains a main lobe and several side lobes. The output of the FIR filter is the baseline signal shown by the solid line in Figure 5.1(a). The longer the filter length (i.e. larger value of L), the narrower the main lobe is and the smoother the filtered output (baseline) will be. The differences between these two sequences are depicted in Figure 5.1(c). The frequency response of the original sequence is shown in Figure 5.2(b). The frequency response of the baseline, i.e. the output of this low-pass filter, is shown in Figure 5.2(c). Figure 5.2(b) and (c), uses log-magnitudes The Matlab®program that generates Figures 5.1 and 5.2 is listed in the Appendix 5.2.

© 2005 by Chapman & Hall/CRC

Table 5.6. Examples of digital filter frequency responses

Filter type	Transfer function	Magnitude frequency response plot
Low pass	$\frac{1-a}{2}\frac{1+z^{-1}}{1-az^{-1}}$, $a = 0.8$	
High pass	$\frac{1+a}{2}\frac{1-z^{-1}}{1-az^{-1}}$, $a = 0.8$	
Band pass	$\frac{1-a}{2}\frac{1+z^{-2}}{1-b(1+a)z^{-1}+az^{-2}}$, $a = 0.8, b = 0.5$	
Band stop	$\frac{1-a}{2}\frac{1-z^{-2}}{1-b(1+a)z^{-1}+az^{-2}}$, $a = 0.8, b = 0.5$	

© 2005 by Chapman & Hall/CRC

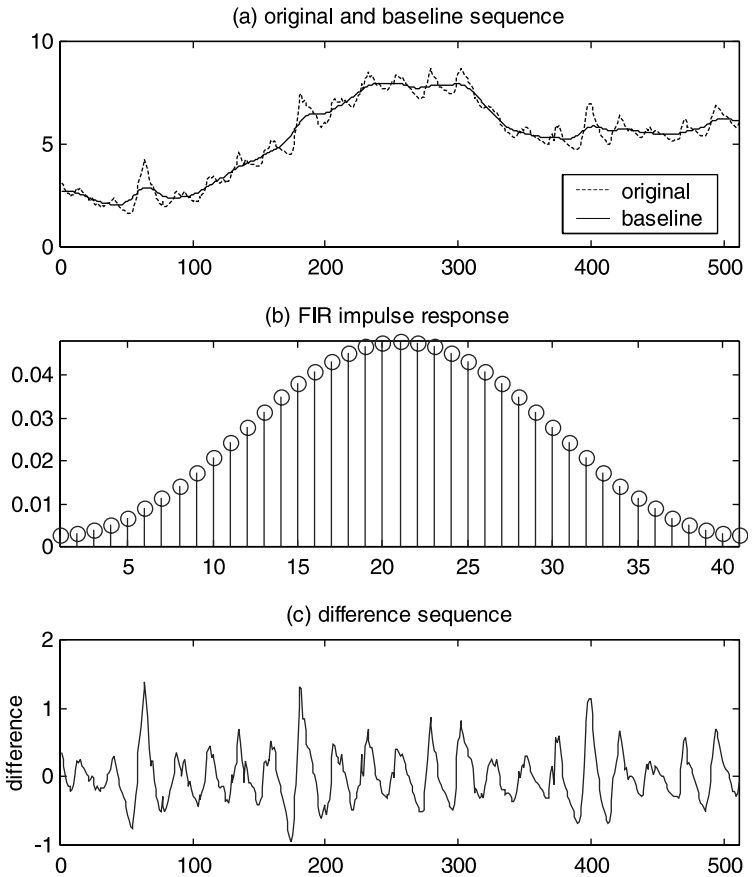


Figure 5.1. Baseline wander removal; (a) original time sequence (dotted line) and baseline wander (solid line), (b) low-pass filter impulse response, and (c) digital sequence with baseline wander removed.

5.5 Sampling, Decimation, Interpolation

5.5.1 Sampling Continuous Analog Signal

An important question in sensor application is how to set the sampling rate. Denote $x(t)$ to be a continuous time signal. When an A/D converter performs sampling, every T seconds, the quantized value of $x(nT) = x(n)$ is obtained. The question then is how much information is lost during this sampling process, and how important is the lost information in terms of reconstructing $x(t)$ from $x(nT)$.

Suppose that the Fourier transform of $x(t)$

$$X(f) = \int_{-\infty}^{\infty} x(t)\, \mathrm{e}^{-\mathrm{j}\omega t}\, \mathrm{d}t \tag{5.7}$$

is such that $|X(f)| = 0$ for $|f| < f_0$, and that the sampling frequency $f_s = 1/T \geq 2f_0$; then, according to the classical Shannon sampling theorem, $x(t)$ can be recovered completely through the interpolation formula

$$\hat{x}(t) = \sum_{n=-\infty}^{\infty} x(n)\, \frac{\sin[\pi(t - nT)/T]}{\pi(t - nT)/T} = \sum_{n=-\infty}^{\infty} x(n)\, \mathrm{sinc}\left(\frac{t - nT}{T}\right) \tag{5.8}$$

© 2005 by Chapman & Hall/CRC

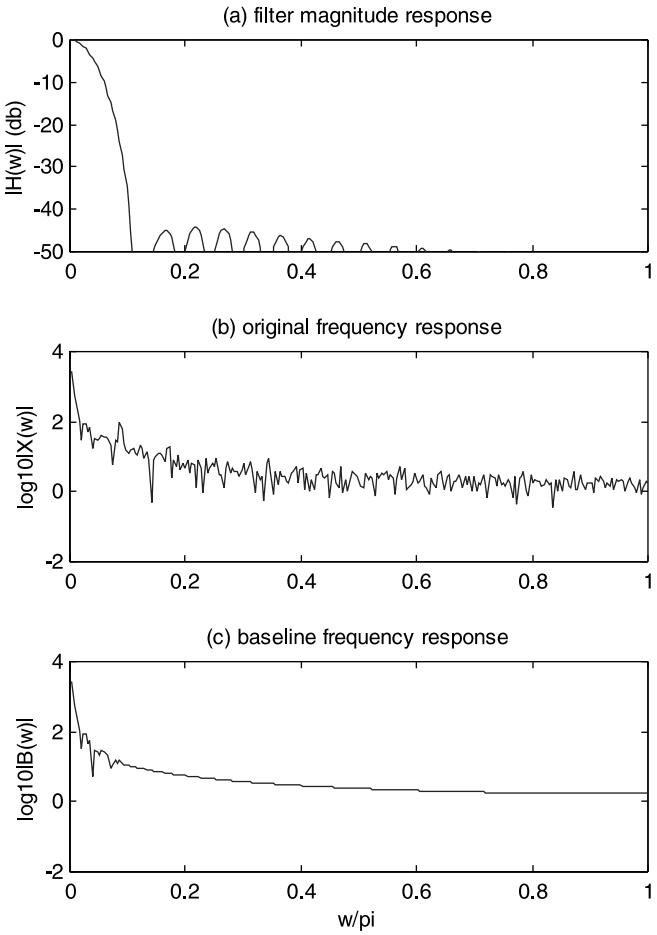


Figure 5.2. (a) Filter frequency response, (b) frequency representation of original sequence, (c) frequency representation of filtered baseline sequence.

where

$$\text{sinc}(t) = \frac{\sin \pi t}{\pi t}$$

In other words, if $x(t)$ is *band limited* with bandwidth f_0, then it can be recovered exactly using Equation (5.8) provided that the sampling frequency is at least twice that of f_0. $f_s = 2f_0$ is known as the Nyquist sampling rate. If the sampling rate is lower than the Nyquist rate, then a phenomenon known as *aliasing* will occur.

Two examples are depicted in Figures 5.3 and 5.4. The first example, in Figure 5.3, shows a continuous time signal

$$x(t) = \cos\left(2\pi f_0 t\right) \quad x \in [0 \quad 1]$$

with $f_0 = 2$ Hz to be sampled at a sampling rate of $f_s = 7$ Hz. The waveform of $x(t)$ is shown as the solid line in the figure. Sampled values are shown in circles. Then, Equation (5.8) is applied to estimate $x(t)$, and the estimated waveform is shown as the dotted line in the same figure. Note that the solid line and

© 2005 by Chapman & Hall/CRC

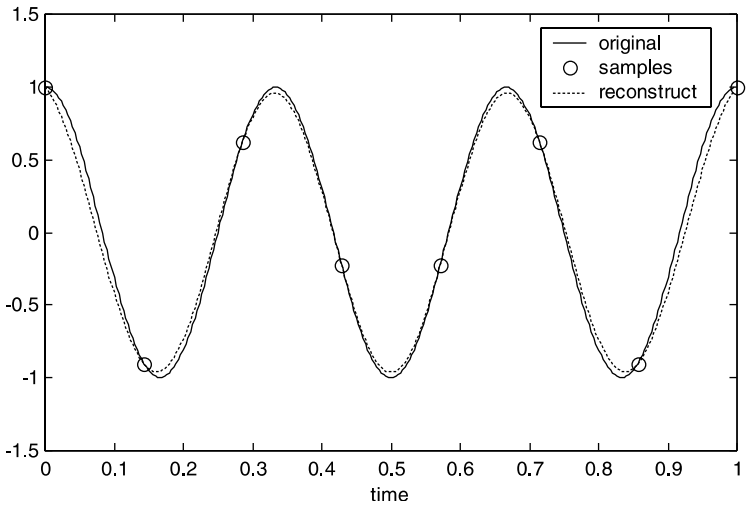


Figure 5.3. Sampling theory demonstration: $f_0 = 2$ Hz, $f_s = 7$ Hz. Solid line is the original signal $x(t)$, circles are sampled data, dotted line is reconstructed signal using Equation (5.8). The mismatch is due to the truncation of $x(t)$ to the interval [0 1].

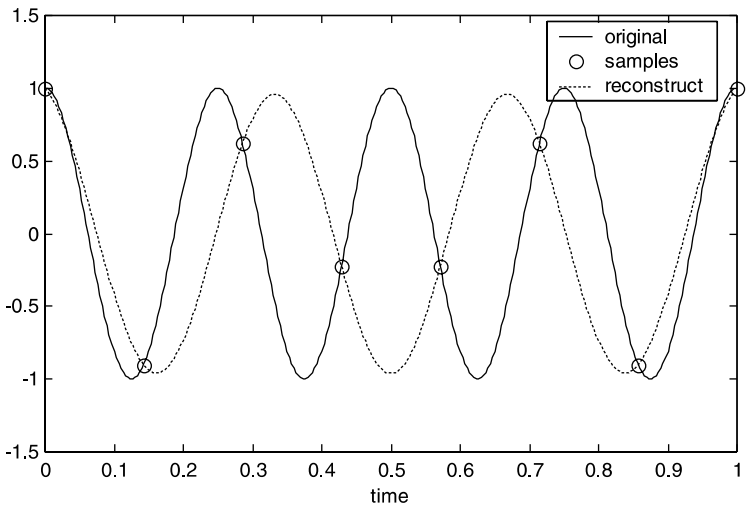


Figure 5.4. Illustration of aliasing effect. $f_0 = 4$ Hz, $f_s = 7$ Hz. Clearly, the reconstructed signal (dotted line) has a lower frequency than the original signal (solid line). Also note that both lines pass through every sampling point. The Matlab® program that generates Figures 5.3 and 5.4 is listed in the Appendix 5.3.

the dotted line do not completely match. This is because $x(t)$ is truncated to the time interval [0 1]. In the second example, in to Figure 5.4, the same $x(t)$ is sampled at a rate that is lower than the Nyquist rate. As a result, the reconstructed curve (dotted line) exhibits a frequency that is lower than the original signal (solid line). Also note that both lines pass through every sampling point.

In practical applications, the bandwidth f_0 of the signal $x(t)$ can sometimes be estimated roughly based on the underlying physical process that generates $x(t)$. It may also depend on which physical phenomenon is to be monitored by the sensor, as well as on the sensor capability and power consumptions. Experiments may be employed to help determine the minimum sampling rate required. One may initially use the highest sampling rate available; then, by analyzing the frequency spectrum of the time series, it is possible to determine the best sampling frequency.

© 2005 by Chapman & Hall/CRC

5.5.2 Sampling Rate Conversion

After a digital signal is sampled, the sampling rate may need to be changed to meet the needs of subsequent processing. For example, sensors of different modalities often require different sampling rates. However, later, during processing, one may want to compare signals of different modality at the same time scale. This would require a change of sampling rate. When the original sampling rate is an integer multiple of the new sampling rate the process is called *down-sampling*. Conversely, when the new sampling rate is an integer multiple of the original sampling rate, the process is called *up-sampling*.

5.5.2.1 Down-Sampling (Decimation)

For a factor of M down-sampling, the DTFT of the resulting signal, $Y(e^{j\omega})$, is related to the DTFT of the original digital signal $X(e^{j\omega})$ by the following expression:

$$Y(e^{j\omega}) = \frac{1}{M}\sum_{k=0}^{M-1} X(e^{j(\omega-2\pi k)/M}) \tag{5.9}$$

If the bandwidth of $X(e^{j\omega})$ is more than $2\pi/M$, then aliasing will occur.

Consider the example depicted in Figure 5.5(a): a signal $x(t)$ is sampled at an interval of $T = 1$ min per sample and 100 samples are obtained over a period of 100 min. The sampled signal is denoted by $x(n)$. The magnitude of the DFT of $x(n)$, denoted by $|X(k)|$, is plotted in Figure 5.5(b). Since $|X(k)| = |X(N-k)|$ $(N = 100)$, only the first $N/2 = 50$ elements are plotted. Note that $|X(k)|$ is a periodic sequence with period f_s, which is equivalent to a normalized frequency of 2π. Hence, the frequency increment between $|X(k)|$ and $|X(k+1)|$ is f_s/N, and the x-axis range is $[0\ f_s/2]$. Note the two peaks at $k = 9$ and 10, representing two harmonic components of period $N/(9f_s) = 11.1$ min per sample and $N/(10f_s) = 10$ min per sample. This roughly coincides with the waveform of $x(n)$ shown in Figure 5.5(a), where a 10 min cycle is clearly visible.

Next, we consider a new sequence by sub-sampling $x(n)$ using a 2:1 ratio. Let us denote this new sequence $y(m) = x(2m+1)$, $0 \le m \le 49$. This is depicted in Figure 5.5(c). Note that the sampling period of $y(m)$ is 2 min per sample. Hence, the time duration of $y(m)$ is still 100 min. Also, note that the sampling frequency for $y(m)$ is now $1/2 = 0.5$ samples per minute. Since $y(m)$ has only 50 samples, there are only 50 harmonic components in its DFT magnitudes $|Y(\ell)|$. These harmonic components spread over the range of $[0\ 2\pi]$, which represents a frequency range of $[0\ 0.5]$ samples per minute. Since we plot only the first 25 of these harmonic components in Figure 5.5(d), the frequency range of the x-axis is $[0\ 0.25]$ samples per minute. Comparing Figure 5.5(d) and (b), the $|Y(\ell)|$ has a shape that is similar to the first 25 harmonics of $|X(k)|$. In reality, they are related as

$$Y(\ell) = \frac{1}{2}[X(\ell) + X(\langle \ell - N/2\rangle_N)] = \begin{cases} [X(\ell) + X(\ell - N/2)]/2 & \ell \ge N/2 \\ [X(\ell) + X(\ell + N/2)]/2 & 0 \le \ell < N/2 \end{cases} \tag{5.10}$$

In this example, the two major harmonic components in $|X(k)|$ have changed very little since they are much larger than other harmonics $\{|X(k)|;\ N/2 \le k \le N-1\}$ shown in hollow circles in Figure 5.5(b). As such, if the sampling rate of this sensor is reduced to half of its original sampling rate, then it will have little effect of identifying the feature of the underlying signal, namely the two major harmonic components. The Matlab® program that generates Figure 5.5 is listed in the Appendix 5.4.

5.5.2.2 Up-Sampling (Interpolation)

With an L-fold up-sampling, a new sequence $x_u(n)$ is constructed from the original digital signal $x(n)$ such that

$$x_u(m) = \begin{cases} x(n) & m = Ln \\ 0 & \text{otherwise} \end{cases}$$

© 2005 by Chapman & Hall/CRC

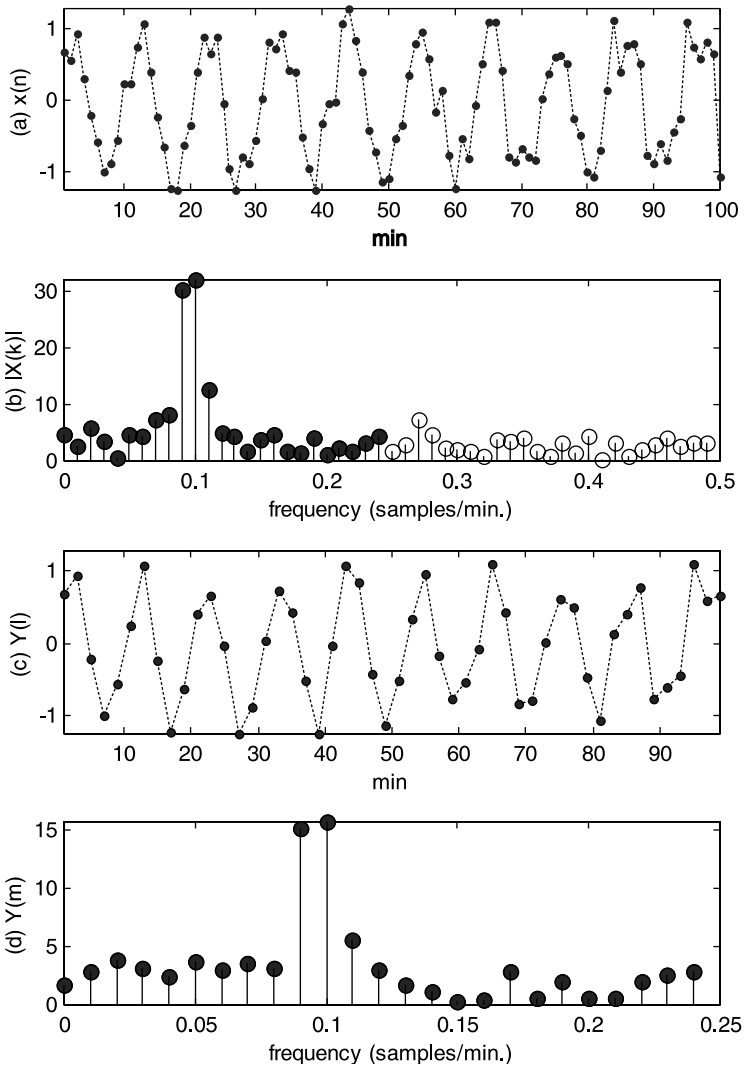


Figure 5.5. (a) $x(n)$, (b) $|X(k)|$, (c) $|Y(\ell)|$, and (d) $y(m)$.

It is easy to verify that

$$X_{\mathrm{u}}(z) = X(z^L)$$

Hence

$$X_{\mathrm{u}}(\mathrm{e}^{\mathrm{j}\omega}) = X(\mathrm{e}^{\mathrm{j}L\omega})$$

and

$$X_{\mathrm{u}}(\ell) = X(\ell \bmod N)$$

However, the zeros in the $x_{\mathrm{u}}(n)$ sequence must be interpolated with more appropriate value in real applications. This can be accomplished by low-pass filtering the $x_{\mathrm{u}}(n)$ sequence so that only one copy of the frequency response $X(k)$ remains.

© 2005 by Chapman & Hall/CRC

5.6 Conclusion

In this chapter, we briefly introduced the basic techniques for the processing of deterministic digital sensor signals. Specifically, the methods of frequency spectrum representation, digital filtering, and sampling are discussed in some detail. However, owing to space limitation, mathematical derivations are omitted. Readers interested in further reading on these topics should check out the many text books concerning digital signal processing, e.g. the three referred to in preparing this chapter.

References

[1] Mitra, S.K., *Digital Signal Processing: A Computer-Based Approach*, McGraw Hill, New York, 2001.
[2] Oppenheim, A.V. and Schafer, R.W., *Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1975.
[3] Mitra, S.K. and Kaiser, J.F., *Handbook for Digital Signal Processing*, John Wiley and Sons, New York, 1993.

Appendix 5.1

Matlab M-file that generates the plots in Table 5.6.

```
% examples of different frequency responses of digital filters
% (C) 2003 by Yu Hen Hu
% examples are taken from Digital Signal Processing, A computer based
% approach, 2nd ed. by S. K. Mitra, McGraw-Hill Irwin, 2001
%
clear all

w=[0:pi/255:pi]; % frequency domain axis
a=0.8;
b=0.5;
% Low pass IIR example
% H1(z)=0.5*(1-a)(1+z^-1)/(1-a*z^-1)
H1=freqz(0.5*(1-a)*[1 1],[1 -a],w);
figure(1),clf
plot(w/pi,abs(H1)),ylabel('|H(w)|'), xlabel('w/pi')
title('(a) Low pass filter')
axis([0 1 0 1])

% High pass IIR
% H2(z)=0.5*(1+a)*(1-z^-1)/(1-a*z^-1)
H2=freqz(0.5*(1+a)*[1-1],[1-a],w);
figure(2),clf
plot(w/pi,abs(H2)),ylabel('|H(w)|'),xlabel('w/pi')
title('(b) High pass filter')
axis([0 1 0 1])

% Band pass IIR
% H3(z)=0.5*(1-a)*(1-z^-2)/(1-b*(1+a)*z^-1+a*z^-2)
H3=freqz(0.5*(1-a)*[1 0 -1],[1 -b*(1+a) a],w);
figure(3), clf
plot(w/pi,abs(H3)),ylabel('|H(w)|'), xlabel('w/pi')
```

© 2005 by Chapman & Hall/CRC

```
title('(c) Band pass filter')
axis([0 1 0 1])

% Band stop IIR
% H4(z)=0.5*(1+a)*(1-2*b*z^-1+z^-2)/(1-b*(1+a)*z^-1+a*z^-2)
H3=freqz(0.5*(1+a)*[1-2*b 1],[1-b*(1+a) a],w);
figure (4),clf
plot(w/pi,abs(H3)),ylabel('|H(w)|'),xlabel('w/pi')
title('(d) Band stop filter')
axis([0 1 0 1])
```

Appendix 5.2

Baseline Wander Removal Example Driver and Subroutine that produces Figures 5.1 and 5.2. Save trendrmv.m into a separate file.

```
% baseline wander removal example
% (C) 2003 by Yu Hen Hu
% call trendrmv.m
% which require signal processing toolbox routines fir1.m, filter.m
% input sequence gt is stored in file try.mat

clear all
load try; % load input sequence variable name gt m x 1 vector
[m,m1]=size(gt);
wcutoff=10/m; % 3dB cut off frequency is set to about 10/512 here
L=input('filter length=2L+1, L=');
[y,ylow,b]=trendrmv(gt,L,wcutoff);
figure(1),clf
subplot(311),plot([1:m],gt, ':',[1:m],ylow, '-'),
legend('original', 'baseline')
title('(a) original and baseline sequence')
axis([0 m 0 10])
subplot(312),stem([1:2*L+1],b),title('FIR impulse response')
axis([1 2*L+1 floor(min(b)) max(b)])
subplot(313),plot([1:m],y),ylabel('difference')
title('(c) difference sequence')
axis([0 m floor(min(y)) ceil(max(y))])

w = [0:pi/255:pi];
Hz=freqz(b,1,w);

figure(2),clf
subplot(311), plot(w/pi, 20*log10(abs(Hz))),
ylabel('|H(w)| (db)')
axis([0 1 -50 0]),title('(a) filter magnitude response')

fgt=abs(fft(gt));
m2=m/2;
subplot(312),plot([1:m2]/m2,log10(fgt(1:m2))),
```

© 2005 by Chapman & Hall/CRC

```
ylabel('log10|X(w)|')
axis([0 1 -2 4]),title('(b) original frequency response')

fylow=abs(fft(ylow));
subplot(313),plot([1:m2]/m2,log10(fylow(1:m2))),
ylabel('log10|B(w)|')
axis([0 1 -2 4]),xlabel('w/pi'),
title('(c) baseline frequency response')

function [y,ylow,b]=trendrmv(x,L,wcutoff)
% Usage: [y,ylow,b]=trendrmv(x,L,wcutoff)
% trend removal using a low pass, symmetric FIR filter
% x is nrecord x N matrix each column is to be low-pass filtered
% L: length of the filter (odd integer > 1)
% wcutoff: normalized frequency, a postive fraction
%                  in terms of normalized frquency
% y: high pass results
% ylow: baseline wander
% b: low pass filter of length 2*L+1
% (C) 2003 by Yu Hen Hu

nrecord=size(x,1);
Npt=2*L; % length of FIR filter
b=fir1(Npt,wcutoff); % low pass filter
% since we want to apply a 2L+1 filter (L=25 here)
% to a sequence of length nrecord
% we need to perform symmetric extension on both ends with L point each
% since matlab filter.m will return an output of same length nrecord+2L
% the output we want are 2L+1:nrecord+2L of the results

temp0=[flipud(x(1:L,:));x;flipud(x(nrecord-L+1:nrecord,:))];
% temp0 is nrecord+2L by nsensor
temp1=filter(b,1,temp0); % temp1 is nrecord+2L by nsensor
ylow=temp1(2*L+1:nrecord+2*L,:);
y=x-ylow;
```

Appendix 5.3

Matlab M-files demonstrating sampling of continuous time signal. Save sinc.m into a separate file.

```
% demonstration of sampling and aliasing
% (c) 2003 by Yu Hen Hu
% call sinc.m

clear all
np=300;
% 1. generate a sinusoid signal
f0=input('Enter freuency in Hertz (cycles/second): ');
tx=[0:np-1]/np; % 512 sampling point within a second, time axis
x=cos(2*pi*f0*tx); % cos(2pi f0 t), original continuous function
```

© 2005 by Chapman & Hall/CRC

```
% 2. Enter sampling frequency
fs=input('Enter sampling frequency in Hertz: ');
T=1/fs; % sampling period
ts=[0:T:1]; % sampling points
xs=cos(2*pi*f0*ts); % x(n)
nts=length(ts);

% 3 computer reconstructed signal.
xhat=zeros(size(tx));
for i=1:nts,
    xhat=xhat+xs(i)*sinc(pi*fs*tx,pi*(i-1));
end
% plot
figure (1),clf
plot(tx,x, 'b-',ts,xs, 'bo',tx,xhat, 'r:');axis([0 1 -1.5 1.5])
legend('original', 'samples''reconstruct')
title(['f_0=' int2str(f0) ' hz, f_s=' int2str(fs) ' hz. '])

function y=sinc(x,a)
% Usage: y=sinc(x,a)
% (C) 2003 by Yu Hen Hu
% y=sin(x-a)/(x-a)
% x: a vector
% a: a constant
% if x=a, y=1;
%

if nargin==1, a=0; end % default, no shift

n=length(x); % length of vector x
y=zeros(size(x));
idx=find(x==a); % sinc(0)=1 needs to be computed separately
if ~isempty(idx),
 y(idx)=1;
 sidx=setdiff([1:n],idx);
 y(sidx)=sin(x(sidx)-a)./(x(sidx)-a);
else
 y=sin(x-a)./(x-a);
end
```

Appendix 5.4

Matlab program to produce Figure 5.5.

```
% demonstration on reading frequency spectrum
% (C) 2003 by Yu Hen Hu
%

clear all
n=100;
f0=0.095; % samples/min
```

© 2005 by Chapman & Hall/CRC

```
%load onem.mat; % variable y(1440,2)
tx=[1:n]'; % sampled at 1min/sample period
tmp=0.2*randn(n,1);
x=sin(2*pi*f0*tx+rand(size(tx)))+tmp;

fs=1; % 1 sample per minute
% spectrum of various length of the sequence
% (a) n point
n2=floor(n/2);
xs=abs(fft(x(:))); %
% dc component not plotted
figure(1), clf
subplot(411),plot([1:n],x(:),'g:',[1:n],x(:),'b.'),
axis([1 n min(x(:)) max(x(:))])
xlabel('min')
ylabel('(a) x(n)')

% (b) subsample 2:1
xc0=x(1:2:n); nc=length(xc0); tc=tx(1:2:n);
xsc=abs(fft(xc0)); nsc=floor(nc/2);

subplot(413),plot(tc,xc0, 'g:',tc,xc0, 'b.')
axis([1 max(tc) min(x(:)) max(x(:))])
xlabel('min')
ylabel(' (c) Y(l)')

tt0=[0:nc-1]/(nc)*(fs/2);
% frequency axis 0 to 2pi, half of samplign frequency
tc=tt0(1:nsc); % plot the first half due to symmetry of mag(DFT)
ltt=length(tc);
subplot(414),stem(tc,xsc(1:nsc), 'filled')
xlabel('frequency (samples/min.)')
axis([0 0.25 0 max(xsc)])
ylabel(' (d) Y(m)')

t2=[0:n2-1]/n2*fs/2;
subplot(412),stem(t2(1:nsc),xs(1:nsc), 'filled'),hold on
stem(t2(nsc+1:n2),xs(nsc+1:n2)),hold off,xlabel('hz')
axis([0 0.5 0 max(xs)])
xlabel('frequency (samples/min.)')
ylabel('(b) |X(k)|')
```

© 2005 by Chapman & Hall/CRC

6

Image-Processing Background

Lynne Grewe and Ben Shahshahani

6.1 Introduction

Images, whether from visible spectrum, photometric cameras or other sensors, are often a key and primary source of data in distributed sensor networks. As such, it is important to understand images and to manipulate them effectively. The common use of images in distributed sensor networks may be because of our own heavy reliance on images as human beings. As the old adage says: ''an image is worth a thousand words.'' Prominence of images in sensor networks may also be due to the fact that there are many kinds of image, not just those formed in the visual spectrum, e.g. infrared (IR), multi-spectral, and sonar images. This chapter will give the reader an understanding of images from creation through manipulation and present applications of images and image processing in sensor networks.

Image processing is simply defined as the manipulation or processing of images. The goal of processing the image is first dependent on how it is used in a sensor network, as well as on the objectives of the network. For example, images may be used as informational data sources in the network or could be used to calibrate or monitor network progress. Image processing can take place at many stages. These stages are commonly called preprocessing, feature or information processing, and postprocessing. Another important image-processing issue is that of compression and transmission. In distributed sensor networks this is particularly important, as images tend to be large in size, causing heavy storage and transmission burdens.

We begin by providing motivation through a few examples of how images are used in sensor network applications. Our technical discussions begin with the topic of image creation; this is followed by a discussion of preprocessing and noise removal. The subsequent sections concentrate on mid-level processing routines and a discussion of the spatial and frequency domain for images. Feature extraction is discussed, and then the image-processing-related issues of registration and calibration are presented. Finally, compression and transmission of images are discussed. Our discussion comes full circle when we discuss some further examples of networks in which images play an integral part.

6.2 Motivation

Before beginning our discussion of image processing, let us look at a few examples of sensor networks where images take a prominent role. This is meant to be motivational, and we will discuss appropriate details of each system in the remaining sections of this chapter.

© 2005 by Chapman & Hall/CRC

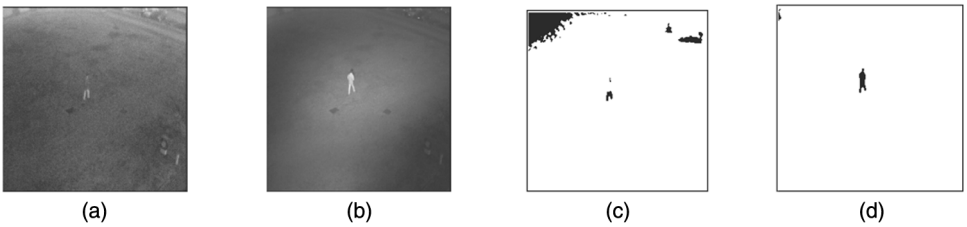


Figure 6.1. Foresti and Snidaro [1] describe a system for outdoor surveillance. (a) Original image. (b) Original IR image. (c) Image (a) after processing to detect blobs. (d) Image (b) after processing to detect blobs.

Foresti and Snidaro [1] describe a distributed sensor network that uses images, both visible spectrum and IR, to do outdoor surveillance. These images are the only source of informational data to find moving targets in the outdoor environment the sensors are monitoring. The use of both kinds of image allows the system to track heat-providing objects, such as humans and other animals, in both day and night conditions. See Figure 6.1 for some images from the system.

A vehicle-mounted sensor network that uses various kinds of image for the task of mine detection in outdoor environments is discussed by Bhatia *et al.* [2]. Here, images play a key role in providing the information to detect the mines in the scene. In particular, IR, ground-penetrating radar(GPR), and metal-detection images are used.

Verma discusses the use by a robotic system of multiple cameras and laser range finders distributed in the work environment to help with goal-oriented tasks that involve object detection, obstacle avoidance, and navigation. Figure 6.2 shows an example environment for this system.

Marcenaro *et al.* [3] describe another surveillance-type application that uses images as the only data source. Here static and mobile cameras are used for gathering data, as shown in Figure 6.3.



Figure 6.2. A distributed sensor environment in which a robot system navigates to attempt to push a box from one location to another. Both cameras and laser range finders are used.

© 2005 by Chapman & Hall/CRC



Figure 6.3. Marcenaro *et al.* [3] developed a system using static and mobile cameras to track outdoor events. Here, a pedestrian is shown trespassing a gate: (a) image from static camera; (b) image from mobile camera.

These are just a few of the many sensor networks that use images. We will revisit these (and others) in the remaining sections as we learn about images and how to manipulate them effectively to achieve the goals of a network system.

6.3 Image Creation

Effective use of images in a sensor network requires understanding of how images are created. There are many "kinds" of image employed in networks. They differ in that different sensors can be used which record fundamentally different information. Sensors can be classified by the spectrum of energy (light) they operate in, the dimensionality of the data produced, and whether they are active or passive.

6.3.1 Image Spectrum

The most commonly used images come from sensors measuring the visible light spectrum. The name of this band of frequencies comes from the fact that it is the range in which humans see. Our common understanding of the images we see is why this is probably the most commonly used imaging sensor in sensor networks. These sensors are commonly called photometric or optical cameras.

The images that we capture with "photometric" cameras measure information in the visible light spectrum. Figure 6.4(a) shows the spectrum of light and where our human-visible spectrum falls. The other images in Figure 6.4 illustrate images created from sensors that sample different parts of this light spectrum.

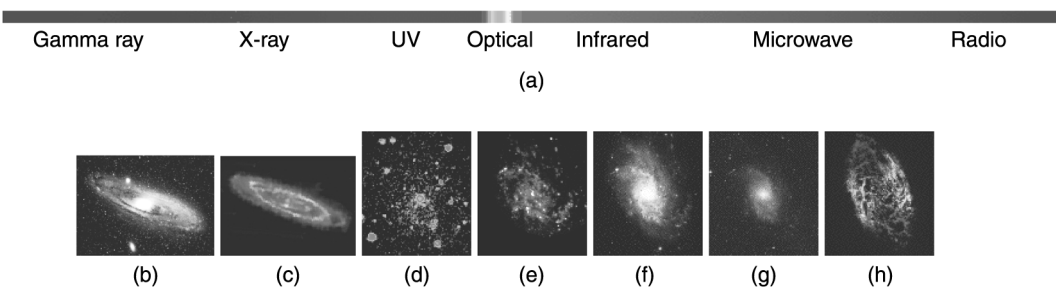


Figure 6.4. Some of the images created via pseudo-coloring image pixel values. (a) Spectrum of light; (b) visible image of the Andromeda galaxy; (c) IR version of (b); (d) x-ray image; (e) ultraviolet (UV) image; (f) visible light image; (g) near-IR image; (h) radio image. (Images courtesy of NASA [4].)

© 2005 by Chapman & Hall/CRC

A sensor also used in network systems is the IR sensor. This sensor has elements that are sensitive to the thermal (IR) region of the spectrum. Forward-looking IR sensors are often used. These involve a passive sensing scheme, which means IR energy emitted from the objects is measured, not that reflected by some source. Near-IR is that portion of the IR spectrum closest to the visible spectrum. Viewing devices that are sensitive to this range are called night-vision devices. This range of the spectrum is important to use when the sensor network needs to operate in no-or low-light conditions, or when information from this band is important (e.g. for sensing heat-emitting objects like animals).

The nearest high-energy neighbor to visible light is the UV region. The sun is a strong emitter of UV radiation, but the Earth's atmosphere shields much of this. UV radiation is used in photosynthesis, and hence this band is used for vegetation detection in images, as seen in many remote-sensing applications.

Some of the other frequency bands are less often employed in sensor network applications, but they are worth mentioning for completeness. The highest energy electromagnetic waves (or photons) are the gamma rays. Many nuclear reactions and interactions result in the emission of gamma rays and they are used in medical applications such as cancer treatments, where focused gamma rays can be used to eliminate malignant cells. Also, other galaxies produce gamma rays thought to be caused by very hot matter falling into a black hole. The Earth's atmosphere shelters us from most of these gamma rays. X-rays, the next band of wavelengths, were discovered by Wilhelm Röentgen, a German physicist who, in 1895, accidentally found these "light" rays when he put a radioactive source in a drawer with some unexposed photographic negatives and found the next day that the film had been exposed. The radioactive source had emitted x-rays and produced bright spots on the film. X-rays, like gamma rays, are used in medical applications, in particular to see inside the body. Finally, microwaves, like radio waves, are used for communications, specifically the transmissions of signals. Microwaves are also a source of heat, as in microwave ovens.

6.3.2 Image Dimensionality

Images can also differ by their dimensionality. All of the images shown so far have two dimensions. However, there are also three-dimensional (3D) images, like the one visualized in Figure 6.5. Here, the information at every point in our image represents the depth from the sensor or some other calibration point in our scene. The term 3D is used because at each point in the image we have information corresponding to the (x, y, z) coordinates of that point, meaning its position in a 3D space. Sensors producing this kind of information are called range sensors. There are a myriad of range sensors, including various forms of radar (active sensors), like sonar and laser range finders, and triangulation-based sensors like stereo and structure light scanners.

Another kind of multi-dimensional image is the multi-spectral or hyper-spectral image. Here, the image is composed of multiple bands (N dimensions), each band has its own two dimensional (2D) image that measures light in that specified frequency band. Multi-spectral images are typically used in remote sensing applications.

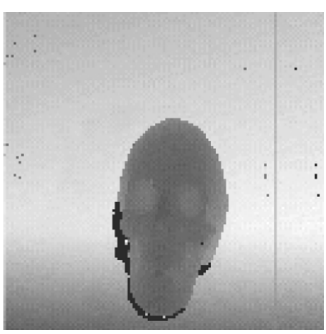


Figure 6.5. Visualization of a 3D image of human skull [5].

© 2005 by Chapman & Hall/CRC

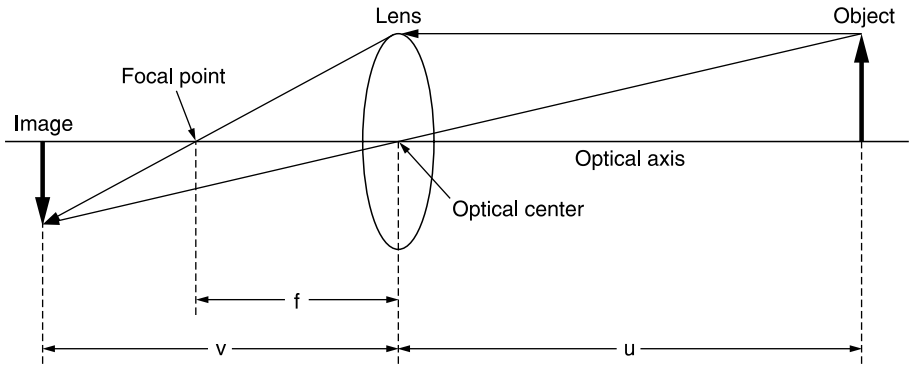


Figure 6.6. Components of a typical camera: sensor plane and lens.

As the most commonly used imaging sensor in network systems is the photometric camera, in Section 2.2.1.4 we will discuss the 2D image structure. However, it is important to note that the image-processing techniques described in this chapter can usually be extended to work with an *N*-dimensional image.

6.3.3 Image Sensor Components

Besides the spectrum or dimensionality, the configuration of the sensor equipment itself will greatly alter the information captured. In the case of photometric cameras, it is made up of two basic components, i.e. a lens and a sensor array. The lens is used to focus the light onto the sensor array. As shown Figure 6.6, what is produced is an upside-down image of the scene.

Photometric cameras are considered passive, meaning that they only register incoming emissions. Other sensors, such as range sensors, are active, meaning that they actively alter the environment by sending out a signal and afterwards measuring the response in the environment. Thus, active sensors will have the additional component of the signal generator.

One example of an active sensor is that of GPR, which is used to detect objects buried underground where traditional photometric cameras cannot see. Here, a signal in a particular frequency range (i.e. 1–6 GHz)is transmitted in a frequency-stepped fashion. Then, an antenna is positioned to receive any signals reflected from objects underground. By scanning the antenna, an image of a 2D spatial area can be created.

Another sensor with different components is that of an electromagnetic induction sensor. This sensor uses coils to detect magnetic fields present in its path. This can be used to detect objects, albeit metallic ones, obscured by the ground or other objects. An image can be comprised by mapping out signals obtained through scanning a 2D spatial area.

There are many ways in which you can alter your environment or configure your sensors to achieve better images for your sensor network application. Some of the performance-influencing factors of a sensor include the dynamic range of the sensor, optical distortions introduced by the sensor, sensor blooming (overly large response to high-intensity signals) and sensor shading (nonuniform response at the outer edges of the sensor array). Lighting, temperature, placement of sensors, focus, and lens settings are some of the factors that can be altered. Whether and how this is done should be in direct relation to the network's objectives. As the focus of this chapter is on image processing, implying that we already have the image, we will not discuss this further. But, it is important to stress how critical these factors are in determining the success of a sensor network system that uses images.

6.3.4 Analog to Digital Images

Figure 6.7 shows the two-step process of creating a digital image from the analog light information hitting a sensor plane. The first step is that of sampling. This involves taking measurements at a specific

© 2005 by Chapman & Hall/CRC

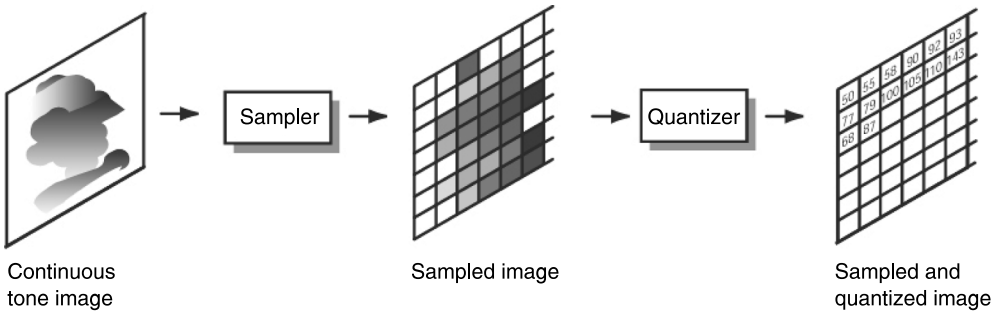


Figure 6.7. Creation of digital image: sampling and quantization.

location in the sensor plane, represented by the location of a sensor array element. These elements are usually distributed in a grid or near-grid pattern. Hence, when we think of a digital image, we often visualize it as shown in Figure 6.7 by a 2D grid of boxes. These boxes are referred to as pixels (picture elements). At this point we can still have a continuous value at each pixel; but, as we wish to store the image inside of a computer, we need to convert it to a discrete value. This process is referred to as quantization.

You lose information in the process of quantization, meaning you cannot invert the process to obtain the original. However, sampling does not have to be a lossy procedure. If you sample at least two times the highest frequency in the analog image, then you will not lose any information.

What results from sampling and quantization is a 2D array of pixels, as illustrated in Figure 6.8. Any pixel in an image is referenced by its row and column location in the 2D array. The upper left-hand corner of the image is usually considered the origin, as shown in the figure.

Through quantization, the range of the values stored in the pixel array can be selected to help achieve the system objectives. However, for the application of display, and for the case of most photometric (visible light) images, we represent the information stored at each pixel as either a grayscale value or a color value.

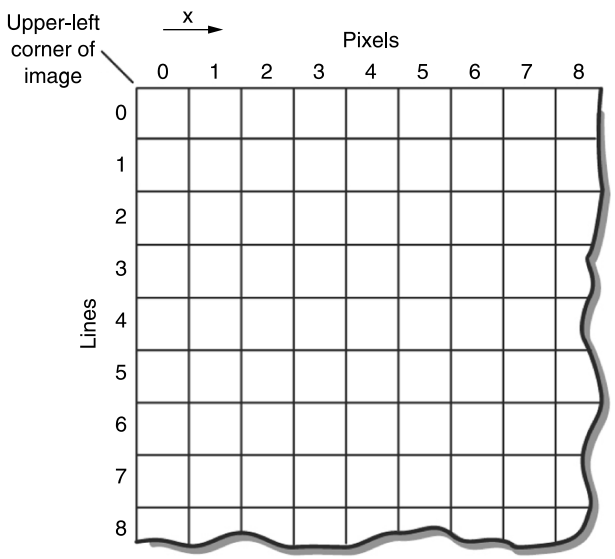


Figure 6.8. The discrete pixel numbering convention.

© 2005 by Chapman & Hall/CRC

In the case of a grayscale image, each pixel has a single value associated with it, which falls in the range of 0 to 255 (thus taking 8 bits). Zero represents black, or the absence of any energy at this pixel location, and 255 represents white, meaning the highest energy the sensor can measure.

Color images, by contrast, typically have three values associated with each pixel, representing red, green, and blue. In today's computers and monitors, this is the most common representation of color. Each color field (red, green, blue) has a range of 0 to 255 (thus taking 8 bits). This kind of color is called 24-bit color or full color and allows us to store approximately 16.7 million different colors. While this may be sufficient for most display applications and many image-processing and-understanding applications it is important to note that there is an entire area of imaging dealing with color science that is actively pursued.

As a note of interest, the difference between a digital and analog sensor is that in a digital sensor the sensor array has its values read directly out into storage. However, analog sensors go through an inefficient digital-to-analog conversion (this is the output of the sensor) and then another analog-to-digital conversion (this time by an external digitizer) sequence before having the information placed in storage.

6.4 Image Domains: Spatial, Frequency and Wavelet

Domains are alternative spaces in which to express the information contained in an image. The process of going from one domain to another is referred to as a transformation. If you can go from one domain to another and return again then the transformation is termed invertible. If you do not lose any information in this transformation process, then the transformation is considered lossless and is called one-to-one. We will discuss the following commonly used image domains: the spatial domain, the frequency domain, and the wavelet domain.

We have already been exposed to the spatial domain; it is the original domain of the image data. The spatial domain is given its name from the fact that neighboring pixels represent spatially adjacent areas in the projected scene. Most image-processing routines operate in the spatial domain. This is a consequence of our intuitive understanding of our physical, spatial world.

The frequency domain is an alternative to expressing the underlying information in the spatial domain in terms of the frequency components in the image data. Frequency measures the spatial variations of the image data. Rapid changes in the pixel values in the spatial domain indicate high-frequency components. Almost-uniform data values mean there are lower frequency components. The frequency domain is used for many image-processing applications, like noise removal, compression, feature extraction, and even convolution-based pattern matching.

There are many transformations that yield different versions of the frequency domain. The most famous and frequently used is the Fourier frequency transformation. The following illustrates the forward and reverse transformation equations, where $f(x, y)$ is the spatial domain array and $F(u, v)$ is the frequency domain array:

$$F(u, v) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) \ \exp\left(\frac{-2\pi j u x}{M}\right) \exp\left(\frac{-2\pi j v y}{N}\right) \tag{6.1}$$

$$f(x, y) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u, v) \ \exp\left(\frac{2\pi j u x}{M}\right) \ \exp\left(\frac{2\pi j v y}{N}\right) \tag{6.2}$$

A fast algorithm Fast Fourier Transform (FFT) is available for computing this transform, providing that N and M are powers of 2. In fact, a 2D FFT transform can be separated into a series of one-dimensional (1D) transforms. In other words, we transform each horizontal line of the image individually to yield an intermediate form in which the horizontal axis is frequency u and the vertical axis is space y.

© 2005 by Chapman & Hall/CRC

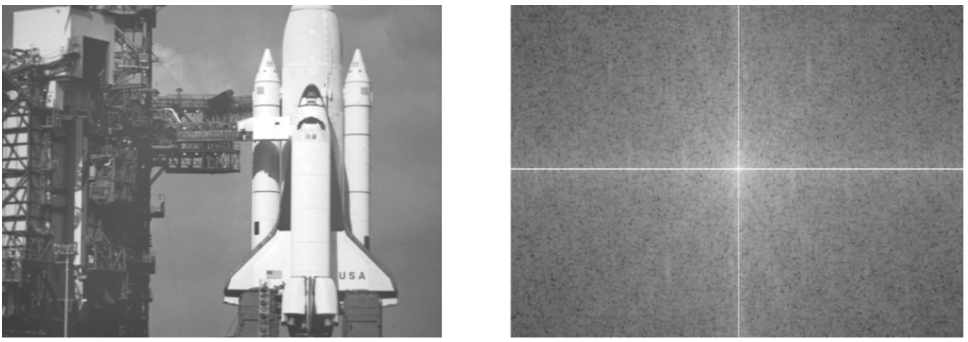


Figure 6.9. Image and its FFT image.



Figure 6.10. Image and its FFT image.

Figures 6.9 and 6.10 show some sample images and their corresponding frequency-domain images. In Figure 6.10(a) we see a very simple image that consists of one frequency component, i.e. of repetitive lines horizontally spaced at equal distances, a sinusoidal brightness pattern. This image has a very simple frequency-domain representation, as shown in Figure 6.10(b). In fact, only three pixels in this frequency domain image have nonzero values. The pixel at the center of the frequency domain represents the DC component, meaning the ''average'' brightness or color in the image. For Figure 6.10(a) this will be some mid-gray value. The other two nonzero pixel values straddling the DC component shown in Figure 6.10(b) are the positive and negative components of a single frequency value (represented by a complex number). We will not go into a discussion of complex variables, but note their presence in Equations (6.1) and (6.2).

The wavelet domain is a more recently developed domain used by some image-processing algorithms. For example, the JPEG standard went from using a discrete cosine transform, another frequency transform similar to the Fourier transform, to using the wavelet transform.

Wavelet basis functions are localized in space and in frequency. This contrasts with a 2D gray-scale image, whose pixels show values at a given location in space, i.e. localized in the spatial domain. It also contrasts with the sine and cosine basis functions of the Fourier transform, which represent a single frequency not localized in space, i.e. localized in the frequency domain. Wavelets describe a limited range of frequencies found in a limited region of space; this gives the wavelet domain many of the positive attributes of both the spatial and frequency domains.

© 2005 by Chapman & Hall/CRC

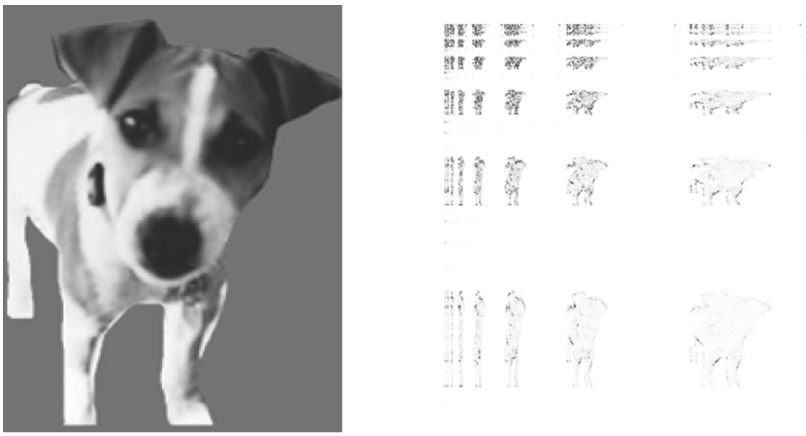


Figure 6.11. Image and its corresponding wavelet domain (visualization of data).

There exist a number of different wavelet basis functions in use. What is common to all of the variations of the wavelet transformation is that the wavelet domain is a hierarchical space where different levels of the transform represent repeated transformations at different scales of the original space. The reader is referred to Brooks *et al.* [6] for more details about the wavelet domain. Figure 6.11 shows an image and the visualization of the corresponding wavelet domain.

6.5 Point-Based Operations

In this section we discuss some of the simplest image-processing algorithms. These include thresholding, conversion, contrast stretching, threshold equalization, inversion, subtraction, averaging, gray-level slicing, and bitplane slicing. What all of these algorithms have in common is that they can be thought of as ''point processes,'' meaning that they operate on one point or pixel at a time. Consider the case of producing a binary image from a grayscale image using thresholding. This is accomplished by comparing each pixel value with the threshold value and consequently setting the pixel value to 0 or 1 (or 255), making it binary.

One way that we can write ''point processes'' is in terms of their transformation function $T()$ as follows: $\text{Pnew}[r, c] = T(P[r, c])$, where $r =$ row, $c =$ column and $P[r, c]$ is the original image's pixel value at r, c. $\text{Pnew}[r, c]$ is the new pixel value.

6.5.1 Thresholding

Thresholding is typically used to create a binary image from a grayscale image. This can be used to highlight areas of potential interest, leading to simple feature extraction and object detection based on brightness information.

This technique can also be used to produce a grayscale image with a reduced range of values or a color image with a reduce range of colors, etc. Notice that in the algorithm below we visit the pixels in a raster scan fashion, meaning one row at a time. This method of visiting all of the pixels in an image is prevalent in many image-processing routines. Figure 6.12(b) shows the results of thresholding a graylevel image.

```
for(r=0; r<M; r++)
for(c=0; c<N; c++)
```

© 2005 by Chapman & Hall/CRC



Figure 6.12. Various point-based operations: (a) original; (b) thresholding image (a); (c) color original; (d) conversion of (c) to grayscale; (e) contrast-stretched version of image (a); (f) histogram-equalized version of image (a); (g) inversion of image (a); (h) bit-plane sliced version of image (a).

```
{
   if(P[r,c] < Threshold)
     Pnew[r,c] = 0;
   else
     Pnew[r,c] = 255;
}
```

6.5.2 Conversion

Conversion is simply converting from one "type" of image to another. Specifically, type here means the modality or dimensionality of the pixel values, such as color or grayscale. For example, consider the conversion of a color image into a grayscale image as performed by the algorithm below. Other types of conversion include converting a full-color image (24-bit/pixel) to an 8-bit color image (using a color LUT). The main application of conversion is to reduce the amount of information associated with an image. If it is possible to perform a task using a reduced set of information, then the network system will run faster and require less energy and storage. Figure 6.12(c) and (d) shows the results of converting a color to a grayscale image.

```
for(r=0; r<M; r++)
for(c=0; c<N; c++)
   Pnew[r,c] = (Red[r,c] + Green[r,c] + Blue[r,c])/3;
```

6.5.3 Contrast Stretching and Histogram Equalization

Contrast stretching and histogram equalization are examples of image-enhancement algorithms that attempt to increase the contrast or range of pixel values present in the image. Low-contrast images can result from poor illumination, lack of dynamic range in the image sensor, or incorrect setting of the lens aperture. The result of a low contrast image is that not all of the possible pixel value range is used, with possibly only a relatively small part of the range being used. By too small, it is meant that the range of values used in the image is not large enough to capture the detail and variation in the original scene clearly. Hence, increasing the contrast of an image can make it easier for both humans and computer programs to extract information for interpretation. Caution should be applied in the use of these algorithms, as their blind use can produce extreme results. This can introduce artifacts, making it more difficult to interpret the images.

Contrast stretching is a simple algorithm that shifts each pixel value by a transformation value that is only a function of the pixel value itself. Commonly, although not required, the transformation can be represented by one or more linear functions. In general, we can define a set of such linear transformations by specifying sets of corresponding pixel values between the original and transformed image that define monotonically increasing lines. Consider Figure 6.13, where we consider a graylevel image and designate two correspondences, p1 and p2. The three linear equations spanning {0,0 to p1},

© 2005 by Chapman & Hall/CRC

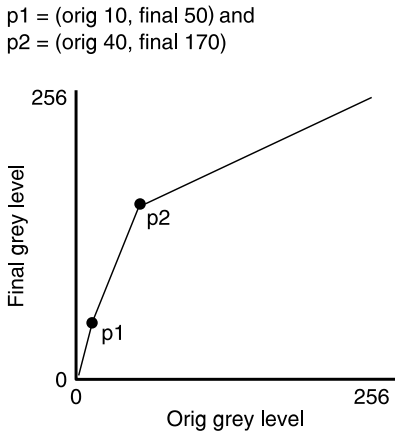


Figure 6.13. Illustration of contrast stretching using set of three linear functions.

{p1 to p2} and {p2 to 256, 256} define the transformation. Figure 6.12(e) shows the results of contrast stretch on the image in Figure 6.12(a).

Histogram equalization is a common technique for enhancing the appearance of images. Suppose we have a grayscale image that is predominantly dark. Its histogram would then be skewed towards the lower end of the grayscale and all the image detail would be compressed into the dark end of the histogram. If we could "stretch out" the gray levels at the dark end to produce a more uniformly distributed histogram then the image would become much clearer. Note, a histogram is simply a count of the number of pixels at each possible pixel value (gray level, color, or whatever).

Unlike in contrast stretching, there are no parameters (i.e. p1, p2) to select. In histogram equalization we are transforming the gray levels such that the resultant image will have a uniform density of gray levels. To achieve this, we use the following transformation.

$$T(x) = (L-1)\sum_{w=0}^{x}\frac{h(w)}{(\#\text{ pixels in image})}$$

where L is the number gray levels in the image and h() is the histogram of the original image. **Note**: $h(w)/(\#\text{pixels in image})$ is the probability of graylevel w occurring in the image.

Figure 6.12(f) shows the results of applying histogram equalization on an image that already has a lot of contrast. This illustrates the fact that histogram equalization should be applied carefully and, in general, only to low-contrast images.

6.5.4 Inversion

This algorithm produces what can be thought of as the negative of the image, as shown in Figure 6.12(g). It is produced by simply inverting the image's pixel range. The application of this algorithm is mostly for data display.

6.5.5 Level Slicing

This is an effect that can be used to highlight a portion of an image's pixel value range. Consider a grayscale image of some coins. There are coins made of silver and pennies made of copper. If you wanted to sort out the pennies then they would correspond to the darker gray circular objects in the image. Hence, you could highlight them in the image by mapping the midlevel gray values to white and

© 2005 by Chapman & Hall/CRC

the rest to black. Applications of this can be found in various object-detection tasks and in multi-spectral remote sensing applications.

6.5.6 Bit-Plane Slicing

Bit-plane slicing is similar to level slicing, but here we examine the bits used to represent each pixel and set certain bits to zero and leave the others untouched. Figure 6.12(h) shows the results of bit-plane slicing on a grayscale image. Some of the higher frequency components can be removed in an image by bit-plane slicing the lower order bits. However, we recommend doing filtering in the frequency domain for this task.

6.5.7 Image Subtraction

Image subtraction is the pixel-by-pixel subtraction of one image from another. One use of image subtraction is the removal of the background. Consider the application of detecting people that come up to an ATM's camera. If you had a picture of the stationary background, and subtracted it from a current image, then only the new items in the scene, e.g. a person, would be visible.

6.5.8 Image Averaging

Image averaging is the pixel-by-pixel averaging of two or more images. Before this process takes place, registration of the images should occur so that averaging only happens between corresponding pixels. One use of image averaging is the reduction of noise in the scene.

6.6 Area-Based Operations

The next class of image-processing algorithms involves looking at a neighborhood of a pixel and using the neighboring pixel values to alter its value. This kind of algorithm can be thought of as an area-processing algorithm. Another term for it is spatial filtering. There are many examples of filters or area-based algorithms in existence, and each can be used for different purposes. In addition to how you combine or use the neighboring pixel values, the size of the neighborhood is another variable. In this section, we will discuss a few of these algorithms.

Let us begin by defining what a neighborhood is. For the pixel labeled (i, j) the neighboring pixels that share borders are highlighted in black in Figure 6.14(a). These pixels are referred to as the four-neighbors of pixel (i, j). All of the pixels surrounding the pixel (i, j) are called eight-neighbors and comprise the smallest neighborhood surrounding this pixel, which is called the 3×3 neighborhood shown in Figure 6.14(b). Increasing the neighborhood size by one row and one column in all directions would yield a 5×5 neighborhood. Typically, most area-based processing algorithms surround a center pixel evenly, which yields an odd number of rows and columns to the processing neighborhood.

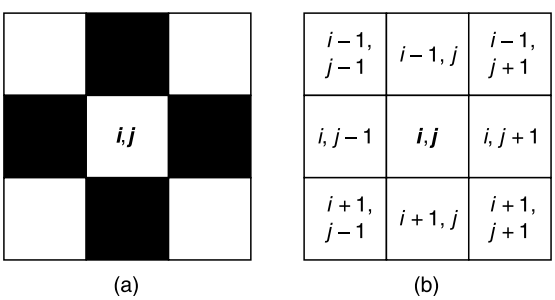


Figure 6.14. (a) Four-neighbors of pixel (i, j); (b) 3×3 neighborhood of pixel (i, j).

© 2005 by Chapman & Hall/CRC



Figure 6.15. Area-based algorithms: (a) original image; (b) low-pass filtered version of image (a); (c) high-pass filtered version of image (a); (d) median filtered version of image (a); (e) detection of horizontal edges; (f) Prewitt edge detection; (g) Sobel edge detection; (h) inversion of magnitude of LOG edge image.

A basic distinction between area-based algorithms is whether they are linear or nonlinear. Linear algorithms replace the center pixel value as a linear function of their neighboring pixels.

6.6.1 Low-Pass Filter

Low-pass filtering is an example of a linear filter, and another name for it is a smoothing filter. This filter is a weighted averaging of the neighboring pixels. This can be represented as convolution with an $N \times N$ (neighborhood size) positively valued convolution mask. A common implementation of a low-pass filter is to set all of the weights to unity and divide the response by a normalization constant if N^*N to keep the image pixel range the same. The following is the algorithm to implement such a smoothing operation. Figure 6.15(b) shows the low-pass filtered image resulting from Figure 6.15(a).

```
for(r=0; r<M; r++)
for(c=0; c<N; c++)
{
   Pnew[r,c] = (P[r-1,c-1] + P[r-1,c] + P[r-1,c+1] +
     P[r,c-1] + P[r,c] + P[r,c+1] +
     P[r+1,c-1] + P[r+1,c] + P[r+1,c+1] )/ 9;
}
```

Low-pass filtering performs smoothing on the image and removes high-frequency details from the image. This is one form of noise removal, and it is also used in an imaging system to remove extraneous details to achieve the system's objectives. Low-pass filtering can be done more efficiently in the frequency domain by simply setting all of the high-frequency components to zero and then taking the inverse transform to obtain the smoothed spatial image.

6.6.2 High-Pass Filter

The high-pass filter performs the opposite function of the low-pass filter; it tries to accentuate the higher frequency information in the image and hence sharpen rather than smooth it. Instead of positive values at neighboring locations in the mask, they are negative. There are numerous kinds of high-pass filter convolution masks, although setting all to -1 is common. High-pass filtering can be done using the frequency domain by adding to the original image a new image formed by setting all but the highest frequencies in the frequency domain to zero and taking the inverse back to the spatial domain. Figure 6.15(c) shows the results of applying a high-pass filter on our bird image.

6.6.3 Median Filter

Median filtering is an example of nonlinear filtering. If the objective is to remove noise and prevent blurring of the image, this approach may be successful. It is particularly useful for removing shot noise,

© 2005 by Chapman & Hall/CRC

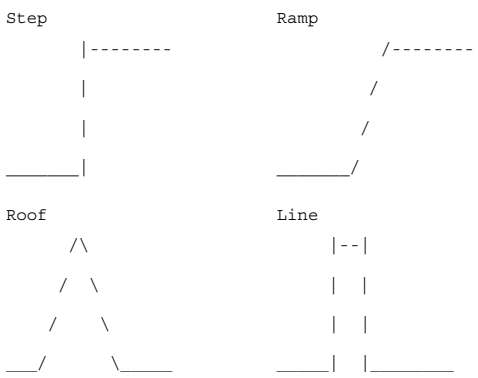


Figure 6.16. Kinds of edge observed in pixel value transctions of a 1D signal.

as discussed in Section 6.7. The algorithm replaces a pixel's value with the median of the pixel values in its neighborhood. Figure 6.15(d) shows the results of applying median filtering to our bird image.

6.6.4 Edge Detection

Edges represent some of the important features of an image. They can describe the boundary of a physical object in the scene or a transition in material or lighting and other environmental effects. The extraction of edge-based features has been used in everything from compression algorithms to, more commonly, image interpretation. Edges can be extracted from many kinds of images, including the visible spectrum, IR, sonar, multi-spectral, and more.

If you were to look at pixel value transitions of a 1D signal (or you can think of this as looking for vertical kinds of edge in a single image row of a 2D image), then the "kinds" of edge that may be observed are shown in Figure 6.16. Of course, such perfect profiles rarely happen, but they illustrate the kind of variation that leads to the observance of an edge point.

There are many ways to detect edges. In the spatial domain, this can be done by convolution with a filter mask, as we describe below. In the frequency domain, one can use high-frequency filters to extract edge information. In either case, what results from the "edge detection" process is simply a new image with the nonzero values representing edge points of varying strength. Extracting features, such as boundaries, lines or other shapes, from an edge-detected image is part of the feature extraction phase and is discussed later in this chapter.

There are many kinds of edge detector, and we will mention a few of the commonly used ones. One of the simplest edge detectors, called the Roberts detector, involves the following 2×2 convolution masks. The edge image is equal to $|Gx| + |Gy|$. Figure 6.15(f) shows the resulting edge image of our bird picture. One unusual thing about this detector is that its masks are not "symmetric," meaning that they do not surround the pixel in question with the same number of pixels in every direction.

Gx =

1	0
0	-1

Gy=

0	1
-1	0

An example of a "symmetric" edge detector is the Sobel, which uses the following masks to produce an edge image equal to the sqrt(Sx*Sx + Sy*Sy). Figure 6.15(g) shows the Sobel edge-detected image for our bird picture. Comparing the results for Roberts and Sobel, you can see that the edge detectors give

© 2005 by Chapman & Hall/CRC

slightly different results. Selecting the right edge detector for your system is often a function of empirical testing.

Sx =

-1	0	1
-2	0	2
-1	0	1

Sy=

1	2	1
0	0	0
-1	-2	-1

The last edge detector we will mention is the LOG or the ''Laplacian of the Gaussian'' operator. This is fundamentally different than the previous two operators, in that they took only a first-order difference whereas the LOG takes the second-order difference (derivative) to measure the presence of an edge. Hence, instead of taking edge points as the maximums of the first-order derivative filters we use the zero-crossings of the second-order derivative. The Laplacian function is the implementation of the second-order derivative. However, first, a Gaussian function is applied to do some smoothing. This two-step procedure is modeled after how our human eyes detect edges. The edge-detected image is produced using the following single convolution mask. Often, the magnitude of the result values is used, with zero values indicating the edge points. Here, we show a 5×5 mask, but larger masks can be employed. Figure 6.15(h) shows the LOG image of our bird picture.

LOG =

0	0	-1	0	0
0	-1	-2	-1	0
-1	-2	16	-2	-1
0	-1	-2	-1	0
0	0	-1	0	0

6.6.5 Morphological Operators

Morphological operators get their name from the fact that they alter the form of the image. Typically, they are applied to images that have been made binary through thresholding or that have been taken into a ''feature-like'' space, such as an edge-detected image. We will discuss dilation and erosion, two commonly used morphological operators.

Dilation is used to expand the boundary of a blob in the image. A blob is any contiguous set of nonzero pixels. In its simplest form, it is used to expand the boundary of blobs by one pixel in each direction. An example of this is shown in Figure 6.17(a). Dilation is useful in joining nearby blobs that were separated erroneously in processing.

Erosion, by contrast, removes pixels at the boundary of a blob. Figure 6.17(b) illustrates this process with an erosion pattern of one pixel. Erosion can be used to reduce a blob to its center location or to trim it down.

Note, any pattern of pixels, rather than simply one pixel, can be used in the erosion or dilation process.

© 2005 by Chapman & Hall/CRC

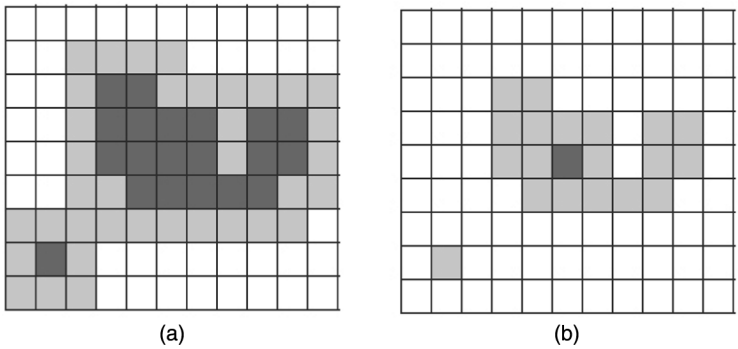


Figure 6.17. Morphological operations: (a) dilation by one pixel, where gray pixels are newly added; (b) erosion by one pixel, where gray pixels are the removed pixels.

6.7 Noise Removal

An image can have noise introduced at many stages, including during creation and transmission. In distributed sensor networks this can be a serious issue, and hence the topic is discussed in a number of the chapters of this book. Note that we differentiate between noise and artifacts or distortions introduced through manipulating the data. Depending on whether the noise is signal-dependent or -independent noise determines how it is handled. Signal-independent noise means that the noise is statistically independent of the image pixel values and hence is treated like an additive component: $I' = I + N$, *where I is the perfect image and N is the noise added resulting in the noisy image* I'. *Noise that is added during transmission of an image is often signal independent.*

Regardless of origin, it is possible to create a measure of the amount of signal-independent noise present at that time. For images, this is commonly done by taking an image, which has a known continuously valued area, and measuring how far the actual image diverges from this known value. This divergence is usually measured by the mean difference and the standard deviation. Another measurement of the significance of the noise is what is called the signal-to-noise ratio and is the ratio of these two magnitudes. This metric is commonly used to describe the performance of a sensor [7].

Signal-dependent noise means that the noise introduced is a function of the original image signal itself. This kind of noise is much more difficult to deal with, because discovering this functional relationship is often not possible with great certainty.

Noise is usually described by its probabilistic characteristics. For example, white noise is given its name because it has a constant power spectrum (its intensity does not decrease with increasing frequency). Gaussian noise has a Gaussian function for its power spectrum. When nothing is known about the origin of a noise signal then it is often assumed to be either Gaussian or white.

There are a number of techniques that have been proposed to reduce the effects of noise. We have already mentioned use of median and low-pass filters for this task. Both can be used for noise reduction. The results on our noisy dog image are shown in Figure 6.18(d) and (e). These are great techniques if not much is known about the noise. Median filtering works best when the noise is ''shot'' noise. This kind of noise occurs sometimes in transmission due to weather disturbances. As you can see, our random noise pattern (which is not shot), is improved the most through the low-pass filter, which simply eliminates the highest frequency patterns. Unfortunately, with any kind of noise-reduction algorithm we may also eliminate or subjugate non-noise information. As we can see in both Figure 6.18(d) and (e), we remove some of the non-noise finest edge, high-frequency information.

Another technique used when we have more than one image of the scene, which is possible in sensor networks, is to average the registered images. As with most stochastic processes, if we sample enough images, then the ensemble mean approaches the noise-free original signal. However, it is not always feasible to have multiple images, and the registration has to be very good not to induce new artifacts.

© 2005 by Chapman & Hall/CRC

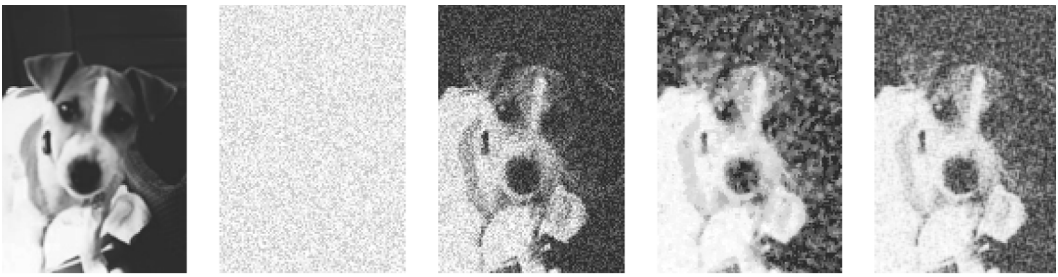


Figure 6.18. (a) Original image; (b) noise image; (c) noisy image; (d) median filter applied to image (c); (e) low-pass filter applied to image (c).

Zhang *et al.* [8] discuss a system for image fusion that uses median filters and neural-network-based fusion for the purpose of image noise reduction. First the images, of the same scene and registered, are preprocessed by a weighted median filter for the purpose of removing noise. Next, an image clustering/segmentation routine is applied using the neural networks. Finally, fusion takes place with the clustered information.

Specific kinds of noise, including environmental and atmospheric-produced noise, are described in subsequent chapters of this book.

6.8 Feature Extraction

In distributed sensor networks that use images, some of the common tasks involve object recognition, navigation, and scene understanding. These high-level tasks must go beyond simply processing images at the pixel level to the extraction of higher-order information. This higher-order information is commonly called "features," and the process to obtain them is called "feature extraction".

There are two basic kinds of feature: statistical and structural (there can be hybrids). Statistical features are those that are measured using statistics. An example might include the average pixel value of an area, the standard deviation, texture, etc. Structural features typically take on a physical or structural makeup. Some examples are edges or boundaries of objects, and surfaces. Of course, you may gather statistics on structural elements, like average number of edges in an area, etc.

We will discuss here a few of the many features that are used in sensor networks. It must be stressed that the features that will yield good results are entirely a function of the network's objectives and the environment. The fact that there is not a definite language to our visual world is what makes image processing and image understanding more challenging than speech recognition.

6.8.1 Edges

Edge following or linking is the process of "chaining" together edge pixels by starting at some edge pixel location and traveling to its neighbor, and so on, until the edge "chain" stops or comes to a junction. This process assumes that there is a definite chain. We discussed previously how to obtain an edge image. What is observed in an edge image is a continuum of edge values. Usually, there are prominent edge pixels, which are surrounded by edge points that to varying degrees are edge points themselves (see Figure 6.19). What we must do before we can run our edge-following algorithm on the image is to reduce down the number of edge pixels to the "essential" ones, meaning in Figure 6.19 to eliminate the gray-colored pixels. This reduction can be achieved through the use of "closing," which involves the sequence of dilation followed by erosion (see Section 6.6.5). Dilation helps close any open boundary areas and erosion reduces the boundary. "Closing" can cause some problems, including reducing the length of a boundary, which could impair recognition.

© 2005 by Chapman & Hall/CRC

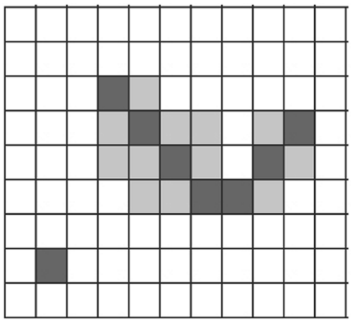


Figure 6.19. Small edge image where the strongest edge points are shown in black and lesser edge points are in gray. These are exactly the pixels eliminated with "thinning."

An alternative is to perform "thinning." Thinning is an algorithm which, unlike "closing," will never lead to removed blobs, reduced boundary lengths, or disconnection of boundaries. Thinning performs erosion, but it will not eliminate a pixel if it is by itself or if it is the only pixel connecting other edge pixels. Thinning would result in removal of the gray-colored pixels in Figure 6.19. After thinning we can easily apply our edge-following algorithm with consistent results.

There are many data structures that are used to store an edge as it is being traced. The most commonly used is the linked list. Once an edge is traced, various attributes can be calculated, like length, curvature, end points, orientation, center, shape, etc.

6.8.2 Hough Transform: Detecting Shapes

The Hough transform is a technique to search for parameterized shapes, like lines or circles, in an image. Although you could try to trace out these shapes using an algorithm like the edge-following algorithm, if the shapes you are looking for are parameterized then the Hough transform is a more efficient way of detecting them.

Like edge following, the Hough transform is applied to an image that has been processed so that the nonzero pixels represent the presence of potential shape points. Often, this may be an edge image, which has been thresholded such that its nonzero pixels represent edges of scene objects, like their boundaries. Consider the simple idea of taking an image of a road scene. You could find the roads by looking for long, straight lines in the image. To do this you would first create a thinned edge image and then you could apply the Hough transform to detect all of the long, straight lines in the image.

For every parameterized shape you are searching for you will need to apply its Hough transform to the image. The basic idea of a Hough transform is that every nonzero pixel (e.g. edge point) votes for all examples of the shape that pass through it. Votes are collected in the "parameter space" of that shape, and the algorithm looks for peaks in this "parameter space." These peaks indicate the strongest instances of the shape in the image.

Let us look at the Hough transform for lines in more detail. A straight line can be described by its perpendicular distance from the origin and its angle from the horizontal axis, as shown in Figure 6.20(a). The parameters r and θ uniquely describe a straight line in an image. The Hough transform parametric space is the 2D space described by the values of r and θ. This space is discretely sampled, and each bin, which acts as a counter, is set to zero. The Hough transform algorithm visits each nonzero pixel in the image (x', y') and determines the curve equation with unknowns (r, θ) and increment all of the (r, θ) cells in the discrete Hough transform space the curve intersects. Figure 6.20(b) shows the bins being increments in the Hough space. As you can see, peaks in the Hough space will be created when a number of pixels coincide on the same line, thus voting for that line.

© 2005 by Chapman & Hall/CRC

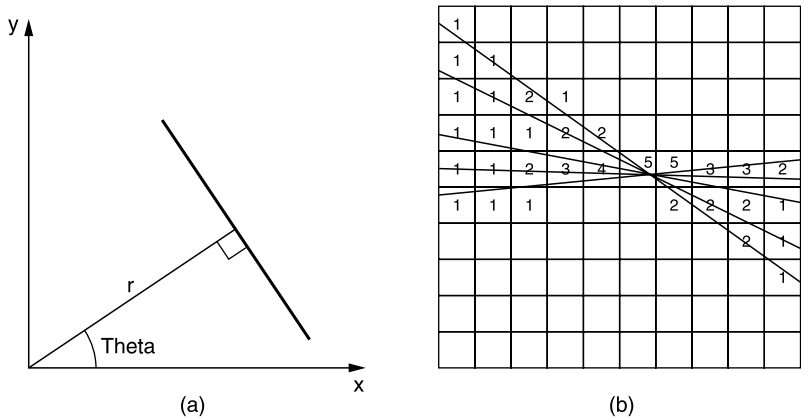


Figure 6.20. (a) A straight line can be described by its parameters r and θ. (b) Hough transform space for a straight line.

6.8.3 Segmentation: Surfaces

Segmentation is the process of breaking up an image into regions that are similar in terms of some property. The goal is usually to divide an image into parts that have a strong correlation with parts of scene objects. Totally correct and complete segmentation of complex scenes cannot usually be achieved. Segmentation works best when you have contrasted objects on a different and ideally uniform background. Common properties used to segment the image include pixel value, statistical measures, and texture.

The simplest way to segment an image is through the process of thresholding. Single or multiple threshold values can be used. Often times, and as a function of the system's environment, various image-processing algorithms will be applied first to remove noise and emphasize the homogeneity of properties within a segment. Figure 6.21 illustrates this process.

More elaborate and better segmentation routines exist, including run-length encoding, split-and-merge and border tracing. The split-and-merge algorithm is a two-step process. First, the algorithm recursively splits an image region (starting with the entire image) by dividing it into quarters. Feature vectors are computed for each block based on some property. If the four blocks are judged to be similar, then they are grouped back together and the process ends. If not, each of the blocks is recursively divided and analyzed in the same fashion. When we are finished with this stage, there will then be many different-sized blocks, each of which has a homogeneous property value. However, the arbitrary division of the image into rectangles (called a ''quad-tree'' decomposition) might have accidentally split up regions of homogeneous property. The second step, i.e. merging, then tries to fix this by

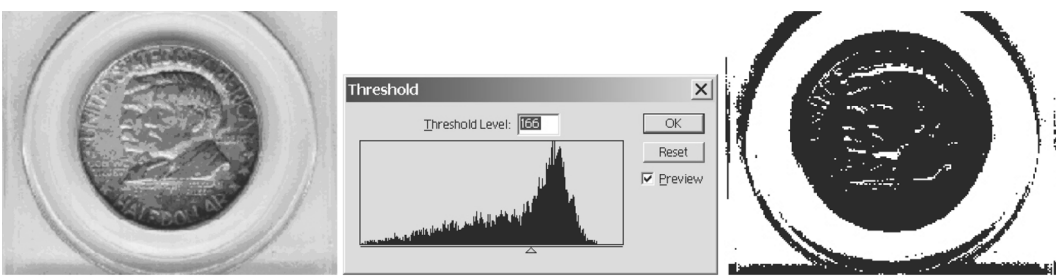


Figure 6.21. (a) Original image. (b) Histogram of brightness values showing thresholding point. (c) Resulting binary image.

© 2005 by Chapman & Hall/CRC

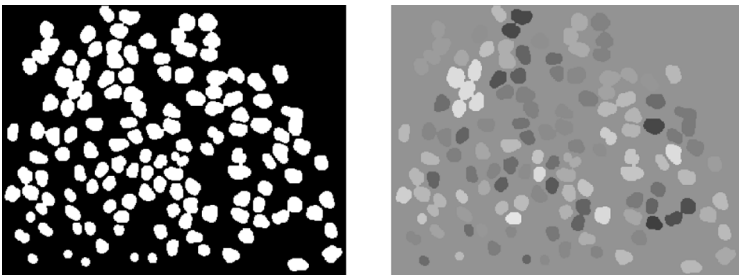


Figure 6.22. (a) Original binary image. (b) Pseudo-colored segmented image; each color shows a different segment detected.

looking at adjacent regions that were not compared with each other during the split phase, and merging them if they are similar. Figure 6.22(b) shows the results of this algorithm on the binary image in Figure 6.22(a).

6.8.4 Examples

A system for feature extraction from multisensorial oceanographic imagery has been discussed by Marcello *et al.* [9]. After an initial preprocessing stage of noise removal is performed, the image is divided up into smaller images called regions of interest (ROIS). The histogram of an ROI is used to threshold it, and the segmentation takes place to find contiguous areas of ''upwelling'' and ''filaments'' in the water images.

Feature extraction can take place in either the spatial or the frequency domain. For example, feature extraction can take place in the wavelet domain using cluster-based (segmentation) techniques [10].

6.9 Registration, Calibration, Fusion Issues

Registration and calibration are steps dealing with understanding the data samples from a sensor source either in relation to other sensor data (registration) or in terms of some absolute scale (calibration). This understanding is critical if sensor networks are to be able to reason over the multiple sensor data acquired.

6.9.1 Registration

When we have multiple sources of data, as is often the case for sensor networks, it is common to want to understand the mapping between one point in a sensor's data to the corresponding point in another sensor's data. This is often critical for networks to be able to use the different sensor data collaboratively to understand the environment. Registration describes this mapping. As discussed earlier, registration is an important topic in distributed network systems, and as such it has an entire chapter devoted to it. Hence, in this section, we will not go over the detail of the registration process; instead, we highlight the kinds of image-processing step that registration systems utilize via discussion of some sample system.

Zamora *et al.* [11] created a system to perform registration for multi-sensor images. Here synthetic aperture radar (SAR) and electro-optic (EO) images are registered for the purpose of fusing them. As these are different kinds of image, registration can be challenging. This system performs preprocessing and feature extraction to assist in this task. In particular, edges are used as features; these are extracted following a preprocessing stage, which includes adaptive clustering segmentation. In the case of the SAR images, first the image is histogram equalized to increase contrast, and then a 3×3 median filter for smoothing is applied. After this the SAR image is thresholded into a binary image. In the case of the EO image, the median filter is applied first and then histogram equalization is applied, followed by binarization. The EO image is first median filtered as the authors felt that there was already a lot of

© 2005 by Chapman & Hall/CRC

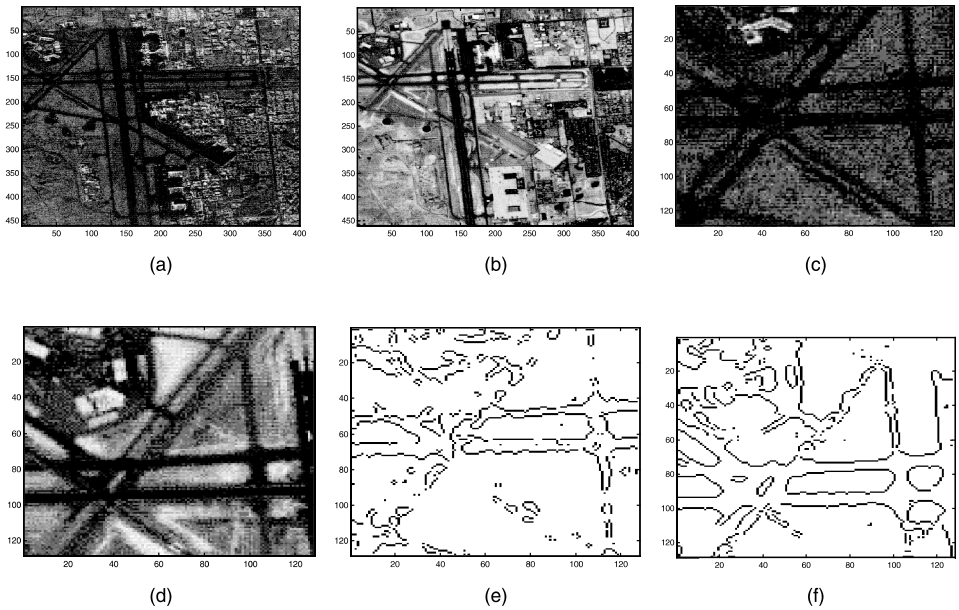


Figure 6.23. System developed for registration of multi-sensor images for the purpose of fusion [11]. (a) Original SAR image; (b) original EO image; (c) clustered SAR image; (d) clustered EO image; (e) edges in processed SAR image; (f) edges in processed EO image.

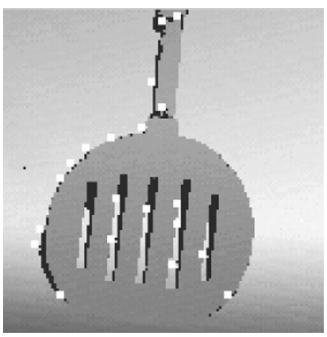


Figure 6.24. Feature points detected from wavelet domain in the 3D image [5].

contrast in the EO images, compared with SAR images, and they wanted to eliminate little unwanted variations. Figure 6.23 shows results on a scene.

Grewe *et al.* [5] presented 3D and 2D photometric images to the system for registration. In this case, a 3D wavelet domain is used to extract features. As described earlier, the wavelet domain is a variation of the frequency domain that captures both frequency and spatial information. In this work, 3D points of inflection are detected in the wavelet domain and used for features to calculate the registration information. Figure 6.24 shows the feature points detected. See the chapter on registration for more details of this and other systems. The reader is also referred to Brown [12] for a survey on image registration techniques.

6.9.2 Geometric Transformations

The product of registration is a transformation matrix that describes the scale, rotation, and translation necessary to transform from one image to another. We discuss each kind of transformation in this section.

© 2005 by Chapman & Hall/CRC

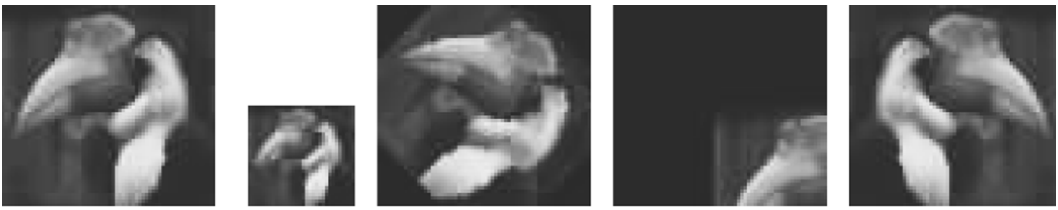


Figure 6.25. (a) Original image. (b) Scaled (shrunk) version. (c) Rotated version. (d) Translated version. (e) Mirrored version around the vertical axis.

Geometric transformations alter the positions of pixels. In other words, the location of a particular pixel value is altered. However, the pixel values also change when the new location falls between pixels, requiring an interpolation process to calculate the new pixel values.

Scaling involves the shrinking or magnification of an image. Shrinking is useful if we need to reduce the number of pixels to be used by our system, because of runtime, energy, or storage restriction considerations. Magnification can be useful when you want to view scenes at larger scale, or even do sub-pixel calculations. Figure 6.25(b) shows a shrunken version of the original bird image in Figure 6.25(a). Suppose we want to shrink our image from $M \times N$ to $M/2 \times N/2$. A simple algorithm to do this would simply average 2×2 blocks of pixels to get a value for each pixel in the new smaller image. Instead, a faster algorithm, but arguably not as good in quality, would be simply to choose one of the pixels in the 2×2 block of the original image as the pixel value for the corresponding pixel in the new smaller image. The first algorithm averages or interpolates the values in a 2×2 block to get a single value. This kind of interpolation is called linear interpolation. There are many kinds of interpolation function that are used, but linear interpolation is the most common.

Rotation of an image means that the laws of trigonometry are applied to rotate the image around its center point by the angle desired. The following equations govern the relationship between the old position (r, c) and the new position (r', c'). Figure 6.25(c) shows a rotated version of our bird image.

$$r' = r\cos(\text{angle}) + c\sin(\text{angle})$$
$$c' = c\cos(\text{angle}) - r\sin(\text{angle})$$

When performing translation, usually parts of images are moved to different locations within the same image. An example is shown in Figure 6.25(d). To specify a translation, you specify the portion of the image and the destination in (row, column) values of this portion. This destination is usually signified as the new location of the upper left corner of the rectangular area to be translated. You may choose to set the original portion of the image not overlapping with the copied version to black, as is done in the image of Figure 6.25(d).

The last geometric transformation we discuss is mirroring. This is the "flipping" of the image around a specified axis or line. An example of this is shown in Figure 6.25(e), where we have mirrored our bird image around the vertical axis.

6.9.3 Calibration

The meaning of sensor calibration depends entirely on the sensor. In the case of visible-spectrum images, camera calibration takes on multiple meanings. The first definition, the most common, is the determination of the parameters that map a position of a point in the scene to its position in the image. The second meaning is that of pixel value calibration, where the dynamic range of the sensor is

© 2005 by Chapman & Hall/CRC

configured to produce expected values. Interestingly enough, this latter meaning is what is typically meant by sensor calibration for nonvisible-spectrum sensors.

Currently, camera calibration is a cumbersome process of estimating the *intrinsic* and *extrinsic* parameters of a camera. There are four intrinsic camera parameters: two are for the position of the origin of the image coordinate frame, and two are for the scale factors of the axes of this frame. There are six extrinsic camera parameters: three are for the position of the center of projection, and three are for the orientation of the image-plane coordinate frame.

Modern CCD cameras are usually capable of a spatial accuracy greater than 1/50 of the pixel size. However, such accuracy is not easily attained due to various errors that can affect the image formation process. Current calibration methods typically assume that the observations are unbiased, that the only error is zero-mean independent and uniformly distributed random noise in the observed image coordinates, and that the camera model completely explains the mapping between the 3D coordinates and the image coordinates. In general, these conditions are not met, causing the calibration results to be less accurate than expected [13].

There are two basic techniques that are commonly employed in camera calibration. The first, called photogrammetric calibration, is done by observing an object whose 3D geometry is known with good precision. The best known such method was performed by TSAI, a system that automatically calibrates a camera, given two planes with a particular shape drawn on them (16 rectangles). The other technique is referred to as self-calibration and is done by moving the camera in a static scene. The rigidity of the scene can be used to produce two constraints on the intrinsic and extrinsic parameters of the camera. Therefore, by obtaining pictures of the camera at different places, we can estimate the intrinsic and extrinsic parameters of the camera.

Calibration can also mean pixel value calibration. Bear [14] discusses color calibration for color cameras. Similar calibration schemes can be developed for each kind of imaging sensor.

6.9.4 Fusion Issues

A number of chapters of this book concentrate on fusion, as this is commonly an important stage of distributed sensor networks. Regarding images and distributed networks, fusion takes place at the pixel level or feature level [15]. When done at the pixel level, the step of registration is especially critical. Both of these types of fusion will be discussed in great detail in later chapters of the book.

6.10 Compression and Transmission: Impacts on a Distributed Sensor Network

Compression of data is important not only for storage, but also for reducing transmission times. This is especially important when the data are large, as typically is the case for images. In distributed sensor networks, it is possible to compress the signal at the sensor before transmission. Compression, however, along with any other processing that might be done at the sensor, is a tradeoff with energy conservation [16]. There have been a number of proposed systems that discuss the development of distributed compression schemes or sensor-based compression schemes for sensor networks, and we will mention a few here.

Martina *et al.* [17] described a system that uses the wavelet transform for compression of images in a wireless sensor network system. Here, to save transmission time, they developed an integrated circuit to implement their algorithm in hardware. While the addition of hardware consumes more power and could be a burden on a distributed sensor network, the savings in transmission time, and hence power, far outweigh this.

Kusuma *et al.* [18] employed a very different approach to the problem. They developed a distributed compression scheme that does not require inter-node communications in the distributed sensor

© 2005 by Chapman & Hall/CRC

network to exploit the correlation. While they show that this scheme can reduce the transmission time, and hence energy consumption, at each sensor node in the network, it is not clear whether this scheme or a hardware-based nondistributed compression scheme like that of Martina *et al.* [17] will yield superior results.

Transmission of information and the related networking issues for a distributed sensor network is a major topic in this book and is discussed in a series of chapters. We will only mention that there is a body of developing work related specifically to images and video and transmission for distributed sensor networks [19].

6.11 More Imaging in Sensor Network Applications

As a closure for our discussion on image processing, we will present a few more examples of sensor networks that use images and image processing. This is only a small sampling of the many that exist, but it will give the reader some idea how the techniques discussed in this chapter are applied.

Baertlein describes a sensor fusion system for detection of buried land mines. Here, the system uses GPR, an IR camera, and an electromagnetic induction sensor. The sensors were used independently and fusion took place via a feature-level fusion scheme that uses an integration of feature information to handle the fact that the sensor data were not necessarily taken at the same locations.

Of interest to us with regard to image processing is the fact that each kind of image was processed differently prior to fusion. Specifically, the GPR images were processed using an iterative technique to try to eliminate ground reflection, a common problem with GPR. Specifically, the onset time and duration of the ground reflection are estimated from low-pass filtered down-range (depth) profiles. The time-domain impulse response of the system is estimated and then iteratively subtracted from the data at points within the ground reflection window, taking care not to remove near-surface targets. The features measured for the GPR image include the cumulative energy. The electromagnetic induction sensor at each measurement point outputs a time-domain waveform. Here, the mean and standard deviation of each trace waveform are used as features. Finally, feature extraction for the IR image is performed via the application of a pattern matching sub-image filter. This filter is circular in structure with a concentric negative outer ring. The result is a binary image containing "blobs" that indicate the potential presence of a mine. A segmentation algorithm is applied, and each feature region is described by its area, pixel count, rectangle dimensions, center, mean temperature (IR measures temperature), and variance.

Yamamoto *et al.* [20] discuss a system that uses images taken from sensors in the air and on ground to assist helicopter pilots in detecting and avoiding obstacles. Both IR and color video camera sensors were used. Each type of image is processed differently before a fusion process takes place. The fused information is used for obstacle detection. In the case of the IR image, a contrast inversion process takes place, followed by sharpening, and finally by histogram equalization. Some of the other filters applied on a case-by-case basis for the IR images included median filtering, to reduce noise, and some thresholding. In the case of the color video images, a geometric transformation takes place to handle registration, followed by conversion to a grayscale image. Figure 6.26 shows some results produced by the system.

A wavelet-domain-based image fusion system for the detection of concealed weapons is discussed by Ramac *et al.* [21]. This system uses IR and millimeter-wave sensors. The images from both sensors are first morphologically filtered to remove image artifacts. These images are then converted to the wavelet domain, where fusion takes place. Figure 6.27 shows some images from this work.

An IR sensor array is used by Feller *et al.* [22] for the purpose of tracking humans for law enforcement applications. They explore the distribution of these sensors geometrically and how this influences real-time recognition and response. The goal is to have the minimal number of sensors and yet be able to view the entire scene area effectively. Registration and geometric transformations are key to understanding this problem.

© 2005 by Chapman & Hall/CRC

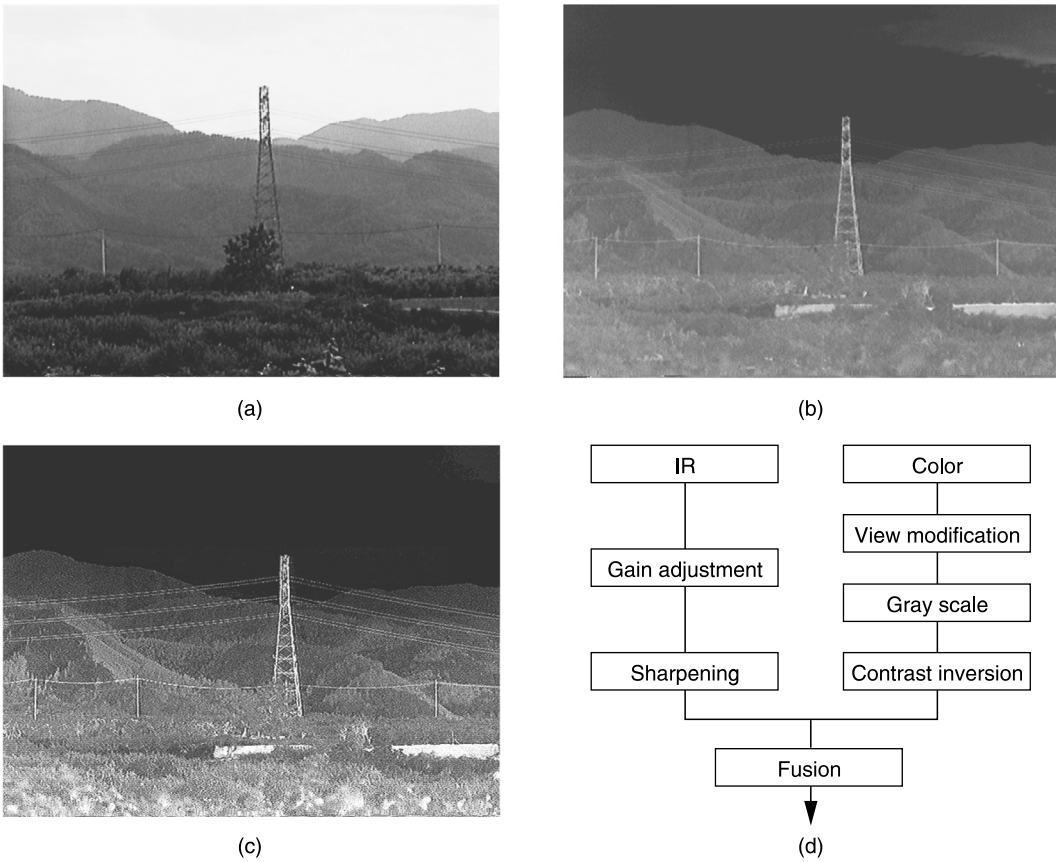


Figure 6.26. System for aiding helicopter pilots [20]. (a) Color video image. (b) IR image. (c) Fused Image. (d) One processing sequence employed by the Yamamoto *et al.* [20] system.

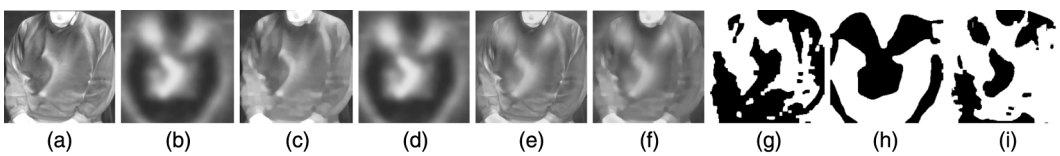


Figure 6.27. Image processing and fusion to detect concealed weapons [21]. (a) Original IR image. (b) Original millimeter-wave image. (c) Image (a) after 5×5 morphological filter applied. (d) Image (b) after 5×5 morphological filter applied. (e) Fused original images. (f) Fused filtered images. (g) Thresholded version image (c). (h) Thresholded version image (d). (i) Fused thresholded images.

References

[1] Foresti, G. and Snidaro, L., A distributed sensor network for video surveillance of for video of outdoor environments, In *International Conference on Image Processing*, 525, 2002.

[2] Bhatia, I. *et al.*, Sensor data fusion for mine detection from a vehicle-mounted system, In *Detection and Remediation Technologies for Mines and Minelike Targets V*, Dubey, A. *et al.* (eds), SPIE Vol. 4038, 824, 2000.

© 2005 by Chapman & Hall/CRC

[3] Marcenaro, L. *et al.*, A multi-resolution outdoor dual camera system for robust video-event metadata extraction, In *FUSION 2002: Proceedings of the Fifth International Conference on Information Fusion*, 1184, 2002.
[4] Multiwavelength Milky Way: across the spectrum, http://adc.gsfc.nasa.gov/mw/mmw_across.html, Version 2, September 2001.
[5] Grewe, L. and Brooks, R.R., Efficient registration in the compressed domain, In *Wavelet Applications VI, SPIE Proceedings*, Vol. 3723, Szu, H. (ed.), Aerosense, Orlando, FL, 1999.
[6] Brooks, R.R., Grewe, L., and Iyengar, S.S., Recognition in the wavelet domain: A survey, *Journal of Electronic Imaging*, 10(3), 757–784, 2001.
[7] Mirzu, M. *et al.*, About the real performance of image intensifier systems for night vision, *Proceedings of SPIE*, 3405, 926, 1998.
[8] Zhang, Z. *et al.*, Image fusion based on median filters and som neural networks: a three-step scheme, *Signal Processing*, 81(6), 1325, 2001.
[9] Marcello, J. *et al.*, Automatic feature extraction from multisensorial oceanographic imagery, In *IEEE International Geoscience and Remote Sensing Symposium*, 2483, 2002.
[10] Sveinsson, J. *et al.*, Cluster-based feature extraction and data fusion in the wavelet domain, In *IEEE International Geoscience and Remote Sensing Symposium*, 867, 2001.
[11] Zamora, G. *et al.*, A robust registration technique for multi-sensor images, In *IEEE Southwest Symposium on Image Analysis and Interpretation*, 87, 1998.
[12] Brown, L., A survey of image registration techniques, *ACM Computing Surveys*, 24(4), 325, 1992.
[13] Zhang, Z., A flexible new technique for camera calibration, http://research.microsoft.com/~zhang/Calib, 2003.
[14] Bear, J., Picture perfection: digital camera calibration, http://desktoppub.about.com/library/weekly/aa072102a.htm, 2003.
[15] Brooks, R.R. and Iyengar, S.S., *Multi-Sensor Fusion: Fundamentals and Applications with Software*, Prentice Hall PTR, Saddle River, NJ, 1998.
[16] Maniezzo, D. *et al.*, Energetic trade-off between computing and communication resource in multimedia surveillance sensor network, In *4th International Workshop on Mobile and Wireless Communications Network*, 373, 2002.
[17] Martina, M. *et al.*, Embedded IWT evaluation in reconfigurable wireless sensor network, In *IEEE International Conference on Electronics Circuits and Systems*, 855, 2002.
[18] Kusuma, J. *et al.*, Distributed compression for sensor networks, In *International Conference on Image Processing*, 82, 2001.
[19] Kostrzewski, A. *et al.*, Visual sensor network based on video streaming and IP-Transparency, In *Battlespace Digitization and Network-Centric Warfare*, SPIE, 4, 2001.
[20] Yamamoto, K. and Yamada, K., Image processing and fusion to detect navigation obstacles, In *SPIE Conference on Signal Processing, Sensor Fusion, and Target Recognition* VII, 337, 1998.
[21] Ramac, L. *et al.*, Morphological filters and wavelet based image fusion for concealed weapons detection, *SPIE* 3375, 110, 1998.
[22] Feller, S. *et al.*, Tracking and imaging humans on heterogeneous infrared sensor arrays for law enforcement applications, In *Sensors, and Command, Control, Communications and Intelligence (C3I) Technologies for the Homeland Defense and Law Enforcement,* SPIE Vol. 4708, 212, 2002.

© 2005 by Chapman & Hall/CRC

7

Object Detection and Classification*

Akbar M. Sayeed

7.1 Introduction

Wireless sensor networks are an emerging technology for monitoring the physical world with a densely distributed network of wireless nodes [1,2]. Each node has limited communication and computation ability and can sense the environment in a variety of modalities, such as acoustic, seismic, and infrared. In principle, sensor networks can be deployed anywhere: on the ground, in the air, or in the water. Once deployed, the nodes have the ability to communicate with each other and configure themselves into a well-connected network. A wide variety of applications are being envisioned for sensor networks, including disaster relief, border monitoring, condition-based machine monitoring, and surveillance in battlefield scenarios. Detection and classification of objects moving through the sensor field is an important task in many applications. Exchange of sensor information between different nodes in the vicinity of the object is necessary for reliable execution of such tasks for a variety of reasons, including limited (local) information gathered by each node, variability in operating conditions, and node failure. Consequently, development of theory and methods for collaborative signal processing (CSP) of the data collected by different nodes is an important research area for realizing the promise of sensor networks.

The exchange of information between nodes for CSP comes at the expense of network resources. The two most critical network resources are: (i) the bandwidth available for communication between nodes, and (ii) the power available at each node for communication and computation. In the case of battery-powered nodes, both constraints are critical. Thus, a key goal in the design of CSP algorithms is to exchange the least amount of data between nodes to attain a desired level of performance. In this chapter, with the above goal in mind, we discuss CSP algorithms for detection and classification of objects using multiple measurements at different nodes.

Some form of region-based signal processing is needed in sensor networks in order to facilitate CSP and also for efficient routing of information through the network, e.g. see [3]. A region-based approach

*This work was supported in part by DARPA SensIT program under grant F30602-00-2-0555.

© 2005 by Chapman & Hall/CRC

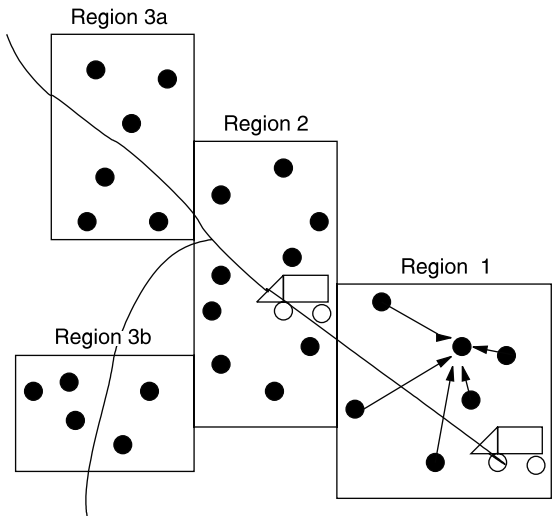


Figure 7.1. A region-based approach for object tracking in sensor networks.

for object tracking is illustrated in Figure 7.1. Typically, the nodes in the network are partitioned into a number of regions and a manager node is designated within each region to facilitate CSP between the nodes in the region and for communication of information from one region to another. Object detection, classification, and tracking generally involves the following steps [3]:

1. **Object detection and data collection.** An object is detected in a particular region which becomes the active region (e.g. region 1 in Figure 7.1). The detection of an object itself may involve CSP between nodes. For example, signal energy measurements at different nodes may be communicated to the manager node to make the final decision about object detection. The nodes within the active region also collect time series data in different modalities that may be used for more sophisticated tasks, such as object classification.
2. **Object localization.** Information related to object detection collected at different nodes (such as the time of closest point of approach and energy measurements) is used by the manager node to estimate the location of the target.
3. **Object location prediction.** Location estimates over a period of time are used by the manager node to predict object location at future time instants.
4. **Creation of new regions.** When the object gets close to exiting the current active region, the estimates of predicted object location are used to put new regions on alert for object detection (e.g. regions 3a and 3b in Figure 7.1).
5. **Determination of new active region.** Once the object is detected in a new region, it becomes the new active region. The above four steps are repeated in the new active region for object tracking through the sensor field.

The CSP techniques discussed in this chapter apply to data collected by different nodes in a particular active region. As we will see, CSP of data collected in different active regions can be done independently. While we specifically discuss CSP methods for detection and classification of a single object, the basic principles apply to distributed decision making in sensor networks in general.

There are two main forms of information exchange between nodes dictated by the statistics of measured signals. If two nodes yield correlated measurements, *data fusion* is needed for optimal performance — exchange of (low-dimensional) feature vectors that yield sufficient information for the desired task. For example, estimates of signal energy at different frequencies (Fourier/spectral feature vectors) may be used for classification. On the other hand, if two nodes yield statistically independent

© 2005 by Chapman & Hall/CRC

measurements, *decision fusion* is sufficient — exchange of soft decisions (real-valued scalars) or hard decisions (discrete-valued scalars) computed at the two nodes. In general, the measurements at different nodes would exhibit a mixture of correlated and independent components and would require a combination of data and decision fusion between nodes. In the context of sensor networks, decision fusion is clearly the more attractive choice. First, it imposes a significantly lower communication burden on the network, compared with data fusion, since only scalars are communicated to the manager node. Second, it also imposes a lower computational burden compared with data fusion, since lower dimensional data have to be jointly processed at the manager node. Third, a classifier based on decision fusion requires a much smaller amount of data for *training*, since fewer parameters characterize the classifier compared with data fusion.

An object in a region covered by a sensor network generates a signal field in space and time that can be sensed by the nodes in different modalities. In Section 7.2 we present a basic but general model for the signal field generated by an object. The model provides a simple characterization of the signal statistics in space and time associated with an object of interest. In particular, the model imposes a universal structure on all CSP algorithms for decision making in which costly[1] data fusion is confined to local sub-regions of an active region, and only cheaper decision fusion is needed across different sub-regions. This model forms the basis of the CSP algorithms presented in the remainder of the chapter. In any network query involving an object (such as a vehicle), the first task is to detect the presence of the object in a region of interest. Section 7.3 discusses CSP algorithms for object detection. Once an object has been detected, the next logical task is to classify the object as belonging to one of a finite number of classes. Section 7.4 discusses CSP algorithms for object classification. In both detection and classification, we discuss algorithms for soft and hard decision fusion and illustrate the performance gains due to multiple node measurements with numerical results. Section 7.5 concludes the chapter with an overview of areas of current and future research.

7.2 A Signal Model for Sensor Measurements

In this section we present a simple model for characterizing the statistics of signals associated with objects in the sensor field. This model will then be used to develop CSP algorithms for object detection and classification. Consider a region of interest $R = D_x \times D_y$, illustrated in Figure 7.2, associated with a network query involving a set of objects. It represents a rectangular region of area $D_x D_y$ (m^2). The network senses the object via the signals collected by the nodes, possibly in multiple modalities. Consider a single sensing modality (e.g. acoustic signals collected by microphones) and a single object (e.g. a vehicle). Let $s(x, y, t)$ denote the signal due to the vehicle at spatial coordinates (x, y) and at time t. The signal $s(x, y, t)$ is best modeled as a random process due to a variety of sources of uncertainty. We assume that $s(x, y, t)$ is zero-mean Gaussian stationary process (or field) as a function of (x, y, t) [4]. As illustrated in Figure 7.2, we divide the region R into spatial coherence (sub-) regions (SCRs) of size $D_{c,x} \times D_{c,y}$ over which $s(x, y, t)$ is assumed constant as function of (x, y), and the constant value varies statistically independently from one SCR to another. The size of an SCR depends on the statistical signal characteristics, as explained next.

To appreciate the notion of SCRs, let us consider a random process $s(t)$ as a function of one variable (time). Since $s(t)$ is a zero-mean Gaussian stationary random process, its statistics are completely determined by the correlation function

$$r_s(\Delta t) = E[s(t)s(t - \Delta t)] = \int_{-B/2}^{B/2} \Phi_s(f) e^{j2\pi f \Delta t}\, df \tag{7.1}$$

which is related to the power spectral density (PSD) $\Phi_s(f)$ of the process via a Fourier transform. B denotes the bandwidth of the process and determines how fast the process changes over time.

[1]This relates to network cost in terms of bandwidth and power expenditure.

© 2005 by Chapman & Hall/CRC

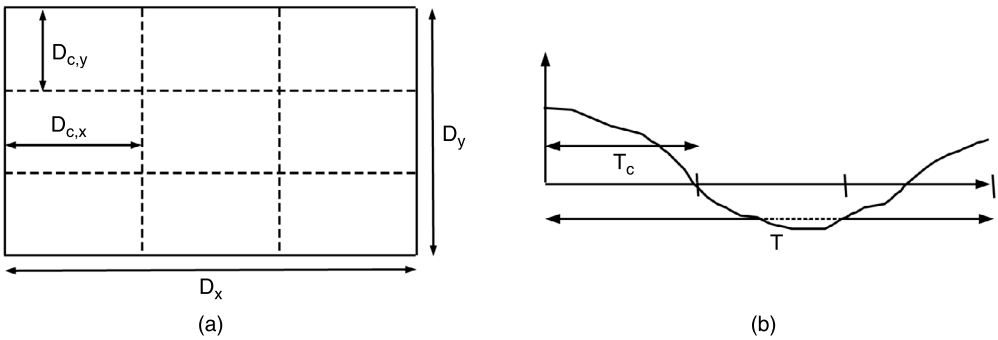


Figure 7.2. A schematic illustrating the notion of spatial coherence regions and coherence time over which the space–time signal of interest $s(x, y, t)$ remains approximately constant (strongly correlated). (a) Coherence regions in space. (b) Coherence intervals in time.

In particular, the process remains strongly correlated over any time interval of duration $T_c = 1/B$, which is called the coherence time and is illustrated in Figure 7.2(b) [5].[2] Thus, all samples of $s(t)$ taken within a duration T_c will be strongly correlated, whereas samples in disjoint time intervals of duration T_c will be approximately statistically independent. The same general principle applies to the space–time signal $s(x, y, t)$. Specifically, let B_x and B_y denote the bandwidths associated with spatial dimensions x and y respectively, analogous to the bandwidth B in time. Then, $D_{c,x} = 1/B_x$ and $D_{c,y} = 1/B_y$ denote the coherence distances in x and y dimensions over which the signal remains strongly correlated. The two coherence distances define SCRs of size $D_{c,x} \times D_{c,y}$, as illustrated in Figure 7.2(a).

7.2.1 Example: Temporal Point Sources

In general, the spatial and temporal signal characteristics can be arbitrary. However, for this important class of signal sources they are coupled. Acoustic signals emitted by vehicles, as well as seismic (vibration) signals produced by moving vehicles, can be modeled in this fashion. Such space–time signals are completely characterized by an underlying temporal signal $s_o(t)$ via signal propagation in space. For isotropic spatial propagation, $s(x, y, t) = s(r, t) = s_o(t - r/v)$ where $r = \sqrt{x^2 + y^2}$ and v is the speed of propagation. Thus, the signal is stationary along radial lines. It can be shown that $B_r = B/v$, where B_r is the spatial bandwidth in the radial dimension and B is the temporal bandwidth of $s_o(t)$. The SCRs are concentric bands around the source and the radial coherence distance $D_{c,r}$ is given by $D_{c,r} = 1/B_r = v/B = vT_c$. For example, for $B = 500\,\text{Hz}$, $D_r = 0.66\,\text{m}$, whereas for $B = 20\,\text{Hz}$, $D_r = 17\,\text{m}$.

Based on the above discussion, we make two slightly idealized assumptions about signal variation as a function of (x, y):

1. $s(x, y, t)$ is perfectly correlated in each SCR; i.e. at any given time the signal in the (i, j)th SCR is constant as a function of (x, y), $s(x, y, t) = s_{i,j}(t)$, $(x, y) \in \text{SCR}_{i,j}$.
2. The signal values $s_{i,j}(t)$ in different SCRs are statistically independent.

In the region $R = D_x \times D_y$, there are $G = N_x N_y$ independent SCRs, where $N_x = D_x/D_{c,x} = D_x B_x$ and $N_y = D_y/D_{c,y} = D_y B_y$, and we label the SCRs as $\text{SCR}_{i,j}, i = 1, \ldots, N_x, \quad j = 1, \ldots, N_y$. In some cases, for simplicity, we will label the SCRs by a single index: $\text{SCR}_k, k = 1, \ldots, G = N_x N_y$. We assume that there are n_G nodes in each SCR, resulting in a total of $K = Gn_G$ nodes in the query region R from which

[2]Note that B is also the required Nyquist rate for sampling the process without any loss of information.

© 2005 by Chapman & Hall/CRC

the measurements are collected. Note that under our model, all n_G nodes in a particular SCR, say $\text{SCR}_{i,j}$ (or $\text{SC}R_k$), observe the same time signal $s_{i,j}(t)$ (or $s_k(t)$).

In practice, the sensor node measurements will be corrupted by noise. Mathematically, the time signal sensed by the kth node is given by

$$z_k(t) = s_k(t) + n_k(t), \quad k = 1, \ldots, K = Gn_G \tag{7.2}$$

where $s_k(t)$ denotes the stationary Gaussian signal due to the object of interest (as discussed above) and $n_k(t)$ denotes additive noise. We assume that $n_k(t)$ is a zero-mean Gaussian white-noise process and that the noise processes at different nodes (whether within an SCR or not) are statistically independent. However, all signal measurements $s_k(t)$ in any particular SCR are identical, and they vary statistically independently from one SCR to the other. At each node, the signal is sampled as

$$z_k[i] = z_k(i/W) = s_k(i/W) + n_k(i/W) = s_k[i] + n_k[i] \tag{7.3}$$

where W denotes the sensor bandwidth (in hertz). We assume that the sampled signal is processed in disjoint blocks of N time samples corresponding to a block duration of $T_o = N/W$ seconds. We denote the nth block of time samples at the kth node as an N-dimensional vector: $\boldsymbol{z}_k[n] = [z_k[nN],\ z_k[nN+1], \ldots, z_k[n(N+1)-1)]^{\mathrm{T}}$. Thus, every T_o seconds, we collect $K = Gn_G$ sampled measurement vectors, $\boldsymbol{z}_k,\ k = 1, \ldots, K$, each of dimension N, and there are n_G vectors in each of the G SCRs.

The above signal model has important implications for distributed detection and classification algorithms, both from a fundamental decision theoretic viewpoint and from the viewpoint of information exchange between nodes. On a fundamental level, there are two sources of error in decision making: (i) the additive noise and (ii) the inherent statistical variability in the source signal. The notion of SCRs illustrated in Figure 7.2 imposes a natural structure on optimal detectors and classifiers that enables us to mitigate both sources of error. Under the assumptions of our signal model, all CSP algorithms for optimal decision making share the following structure:

1. Since the source signal is nearly constant in each SCR, the n_G measurements in each SCR are averaged to mitigate the effect of noise and increase the effective signal-to-noise ratio (SNR) by a factor of n_G.
2. Since the G averaged measurements in different SCRs are statistically independent, they are combined appropriately to reduce the inherent statistical variability in the source signal.

Both of these aspects improve the performance of detection and classification algorithms; but, as we will see, the second effect is more critical in the context of random source signals. For the remainder of the chapter we assume that the n_G sampled vectors $\boldsymbol{z}_k$ in each SCR are averaged to yield a single N-dimensional vector $\boldsymbol{z}_k$ for each SCR:

$$\boldsymbol{z}_k = \frac{1}{n_G} \sum_{i \in SCR_k} \boldsymbol{z}_i = \boldsymbol{s}_k + \boldsymbol{n}_k, \quad k = 1, \ldots, G \tag{7.4}$$

Note that this averaging corresponds to data fusion, since the N-dimensional feature vectors are exchanged between the nodes in each SCR. This data fusion in each SCR could be coordinated by the manager for the entire region R or separate manager nodes could be designated for each SCR. The net result of this averaging is that the signal component remains unchanged, since it is constant in each SCR, whereas the averaging of the noise reduces its variance by a factor of n_G. Thus, if the original noise variance is σ_n^2, then the variance of the averaged noise becomes σ_n^2/n_G. Thus, we work with a total of $G = N_x N_y$ statistically independent averaged measurement vectors $\{\boldsymbol{z}_k\}$ from different SCRs as in

© 2005 by Chapman & Hall/CRC

Equation (7.4), where $\boldsymbol{s}_k \sim \mathcal{N}(\boldsymbol{0}, \Sigma)$ and $\boldsymbol{n}_k \sim \mathcal{N}(\boldsymbol{0}, \sigma_n^2 \boldsymbol{I}/n_G)$. The notation $\boldsymbol{s} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ means that the vector $\boldsymbol{s}$ is a vector of Gaussian (normal) random variables with mean $\boldsymbol{\mu}$ and covariance matrix Σ, i.e.

$$\mu[i] = E[s[i]] \quad \text{and} \quad \Sigma[i,j] = E[(s[i] - \mu[i])(s[j] - \mu[j])], \quad i,j = 1, \ldots, N \tag{7.5}$$

$\boldsymbol{I}$ denotes an identity matrix and $\boldsymbol{n}_k \sim \mathcal{N}(\boldsymbol{0}, \sigma_n^2 \boldsymbol{I}/n_G)$ means that the noise vector has zero-mean independent components with variance σ_n^2/n_G.

7.3 Object Detection

Consider a network query of the form: ''Is there a vehicle in the region R''? This corresponds to object detection, which is a natural precursor to classification of the object involved. Mathematically, this corresponds to a binary hypothesis test

$$H_0: \boldsymbol{z}_k = \boldsymbol{n}_k, \quad k = 1, \ldots, G \tag{7.6}$$

$$H_1: \boldsymbol{z}_k = \boldsymbol{s}_k + \boldsymbol{n}_k, \quad k = 1, \ldots, G \tag{7.7}$$

where $\boldsymbol{z}_k$ denotes the averaged N-dimensional sampled vector from the kth SCR in a given time block (we ignore the block index for simplicity). H_0 corresponds to the hypothesis that no object is present, i.e. $\boldsymbol{z}_k = \boldsymbol{n}_k \sim \mathcal{N}(\boldsymbol{0}, \sigma_n^2 \boldsymbol{I}/n_G)$. On the other hand, H_1 represents the presence of an object, i.e. $\boldsymbol{z}_k = \boldsymbol{s}_k + \boldsymbol{n}_k \sim \mathcal{N}(\boldsymbol{0}, (\sigma_s^2 + \sigma_n^2/n_G)\boldsymbol{I})$. For simplicity we have assumed that the different components of the signal vector $\boldsymbol{s}_k$ are independently identically distributed (i.i.d.) with variance σ_s^2. However, this assumption does not alter the nature of the detector. We consider two types of detector. First, we consider a detector that combines real-valued *soft decisions* from different SCRs and serves as an idealized detector. Second, we consider a practical detector that combines binary-valued *hard decisions* from the G SCRs to make the final decision.

7.3.1 Soft Decision Fusion

Under the above assumptions, the optimal detection statistic is given by

$$l(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_G) = \frac{1}{NG} \sum_{k=1}^{G} \|\boldsymbol{z}_k\|^2 = \frac{1}{NG} \sum_{k=1}^{G} \sum_{n=1}^{N} z_k^2[n] \tag{7.8}$$

which is the average energy in the measurements. Note that in this case each node communicates the energy in its local measurement, $\|\boldsymbol{z}_k\|^2$, to the manager node. The final detector implemented at the manager node is called an *energy detector* and it makes the decision d by comparing l with a threshold γ

$$d(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_K) = \begin{cases} 1 & l > \gamma \\ 0 & l \leq \gamma \end{cases} \tag{7.9}$$

The decision $d = 1$ corresponds to H_1 (object present) and $d = 0$ corresponds to H_0 (no object present). The threshold γ has to be chosen carefully to control detector performance. Two important performance criteria are: (i) the probability of false alarm (PFA) — the probability of declaring an event detection under H_0 (when only noise is present); and (ii) probability of detection (PD) — the probability of declaring an event detection under H_1 (when the signal is actually present). Mathematically:

$$\text{PFA} = P(D = 1|H_0) = P(L > \gamma|H_0), \quad \text{PD} = P(D = 1|H_1) = P(L > \gamma|H_1) \tag{7.10}$$

Note that uppercase L denotes the random variable corresponding to the detection statistic l (lower case) in Equation (7.8). Similarly, uppercase D denotes the random variable corresponding to the decision d (lower case) in Equation (7.9). This customary notation for representing random variables with uppercase letters, and their particular values (realizations) with lowercase letters is used throughout the paper.

© 2005 by Chapman & Hall/CRC

Ideally, we would like PFA to be as small as possible and PD to be as large as possible. In practice, the threshold γ is chosen to keep PFA below a prescribed value (e.g. less than 5%). Let $\sigma_0^2 = E_0[Z_k^2[n]] = E[N_k^2[n]] = \sigma_n^2/n_G$ denote the average power in the measurements under H_0, where $E_0[\cdot]$ denotes expectation under H_0. Similarly, let $\sigma_1^2 = E_1[Z_k^2[n]] = E[S_k^2[n]] + E[N_k^2[n]] = \sigma_s^2 + \sigma_0^2 = \sigma_s^2 + \sigma_n^2/n_G$ denote the average measurement power under H_1, where σ_s^2 denotes the power in each signal vector component. Then, for any even value of NG ($NG = 2m$), the PFA and PD can be computed as a function of $\gamma > 0$ as [5]

$$\text{PFA}(\gamma) = \mathrm{e}^{-NG\gamma/2\sigma_0^2} \sum_{k=0}^{m-1} \frac{1}{k!} \left(\frac{NG\gamma}{2\sigma_0^2}\right)^k \tag{7.11}$$

$$\text{PD}(\gamma) = \mathrm{e}^{-NG\gamma/2\sigma_1^2} \sum_{k=0}^{m-1} \frac{1}{k!} \left(\frac{NG\gamma}{2\sigma_1^2}\right)^k \tag{7.12}$$

Note that PD and PFA depend on N and G as well as the SNR

$$\text{SNR} = \frac{E_1[Z_k^2[n]]}{E_0[Z_k^2[n]]} = \frac{\sigma_1^2}{\sigma_0^2} = 1 + n_G \frac{\sigma_s^2}{\sigma_n^2} = 1 + n_G \text{SNR}_o \tag{7.13}$$

which clearly shows the improvement in SNR as a function of the number n_G of sensor measurements in each SCR (SNR_o denotes the SNR at each sensor).

A typical way of characterizing detector performance is to plot PD(γ) as a function of PFA(γ) for different values of γ. The resulting plot is called the receiver operating characteristic (ROC). Figure 7.3 shows ROC curves for different values of NG. Figure 7.3(a) corresponds to SNR = 2 and Figure 7.3(b) corresponds to SNR = 4. Comparing the two plots, it is clear that the performance improves with SNR. Furthermore, for a given SNR, the performance improves with increasing NG, which could be increased by increasing G — number of independent measurements from different SCRs.

Detector design thus boils down to choosing the value of γ that corresponds to the desired operating point on the ROC. A value of γ between σ_0^2 and σ_1^2 usually suffices. To see this, note that the detection statistic random variable L in Equation (7.8) is the average of NG i.i.d. random variables $\{Z_k^2[n]\}$. Furthermore, since $\{Z_k[n]\}$ are zero-mean Gaussian, by the moment theorem for Gaussian random variables [5] we have

$$E_i[Z_k^2[n]] = \sigma_i^2, \quad \text{var}_i[Z_k^2[n]] = 2\sigma_i^4, \quad i = 0, 1 \tag{7.14}$$

where $\text{var}[Z] = E[Z^2] - (E[Z])^2$ denotes the variance of Z. Consequently, by the law of large numbers [5] we have

$$E_i[L] = \sigma_i^2, \quad \text{var}_i[L] = 2\sigma_i^4/NG, \quad i = 0, 1 \tag{7.15}$$

The mean of L is σ_i^2 under H_i, and thus a value of γ between the two means usually suffices. Furthermore, since the variance of L goes to zero as NG increases (independent measurements reduce the statistical variability in the decision statistic), we expect PFA $\to 0$ and PD $\to 1$ as $G \to \infty$. To see this, let $\gamma = \sigma_0^2 + \delta$, where $0 < \delta < \sigma_1^2 - \sigma_0^2 = \sigma_s^2$. Then, using Tchebyshev's inequality [5], it can be shown that

$$\text{PFA}(\gamma) \le P\left(|L - \sigma_0^2|^2 \ge \delta^2 | H_0\right) \le \frac{\text{var}_0[L]}{\delta^2} = \frac{2\sigma_0^4}{NG\delta^2} \tag{7.16}$$

$$\text{PD}(\gamma) \ge P\left(|L - \sigma_1^2|^2 \le (\sigma_s^2 - \delta)^2 | H_1\right) \ge 1 - \frac{\text{var}_1[L]}{(\sigma_s^2 - \delta)^2} = 1 - \frac{2\sigma_1^4}{NG(\sigma_s^2 - \delta)^2} \tag{7.17}$$

© 2005 by Chapman & Hall/CRC

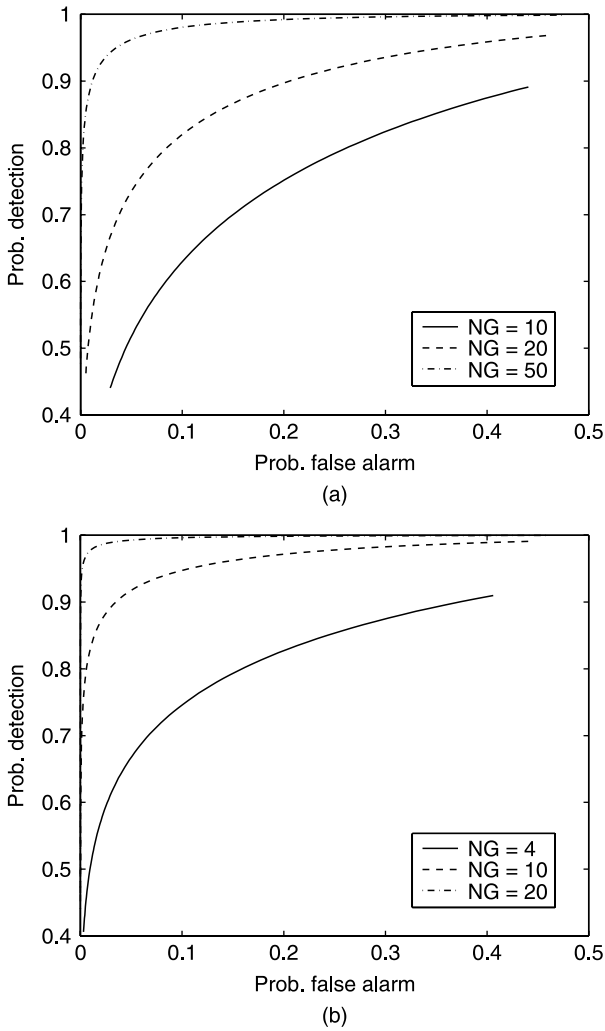


Figure 7.3. ROC curves for the soft decision fusion energy detector for different values of independent measurements NG. The curve for each value of NG is generated by varying γ between σ_0^2 and σ_1^2. (a) $\sigma_0^2 = 1$ and $\sigma_1^2 = 2$ (b) $\sigma_0^2 = 1$ and $\sigma_1^2 = 4$.

Thus, for any value of γ satisfying $\sigma_0^2 < \gamma < \sigma_1^2 = \sigma_0^2 + \sigma_s^2$ we get closer to perfect performance (PD = 1 and PFA = 0) as NG increases.

7.3.2 Hard Decision Fusion

Recall that the measurement energy at the kth SCR, $\|\boldsymbol{z}_k\|^2$, is communicated to the manager node that implements the soft decision fusion detector based on the G energy values. However, energy is a nonnegative real number and, in general, each SCR has to encode it with a finite number of bits (quantized) to communicate it digitally to the manager node. In this section we consider a particular form of quantization in which the kth SCR makes a hard decision based on its local measurement vector $\boldsymbol{z}_k$

$$u_k(\boldsymbol{z}_k) = \begin{cases} 1 & \text{if } \|\boldsymbol{z}_k\|^2/N > \gamma \\ 0 & \text{if } \|\boldsymbol{z}_k\|^2/N \leq \gamma \end{cases} \tag{7.18}$$

© 2005 by Chapman & Hall/CRC

and then communicates its binary decision u_k to the manager node. The final detector at the manager node (for the entire region) computes the following averaged hard decision statistic

$$l'(u_1, \ldots, u_G) = \frac{1}{G}\sum_{k=1}^{G} u_k \tag{7.19}$$

and compares it to a threshold γ' to make the final decision

$$d_{\text{hard}}(u_1, \ldots, u_K) = \begin{cases} 1 & \text{if } l' > \gamma' \\ 0 & \text{if } l' \le \gamma' \end{cases} \tag{7.20}$$

Since the $\{\boldsymbol{z}_k\}$'s are i.i.d. so are $\{u_k\}$. Each u_k is a binary-valued Bernoulli random variable characterized by the following two probabilities under the two hypotheses:

$$p_1[1] = P(U_k = 1|H_1), \quad p_1[0] = P(U_k = 0|H_1) = 1 - p_1[1] \tag{7.21}$$
$$p_0[1] = P(U_k = 1|H_0), \quad p_0[0] = P(U_k = 0|H_0) = 1 - p_0[1] \tag{7.22}$$

We note from Equations (7.18) and (7.8) that $p_1[1]$ is the PD and $p_0[1]$ is the PFA for the soft decision fusion detector when $G = 1$. It follows that the hard decision statistic in Equation (7.19) is a (scaled) binomial random variable under both hypotheses, and thus the PD and PFA corresponding to d_{hard} can be computed as a function of γ' as [5]

$$\text{PD}(\gamma') = P(D_{\text{hard}} > \gamma'|H_1) = 1 - \sum_{k=0}^{\lfloor \gamma' G \rfloor} \binom{G}{k} p_1[1]^k (1 - p_1[1])^{G-k} \tag{7.23}$$

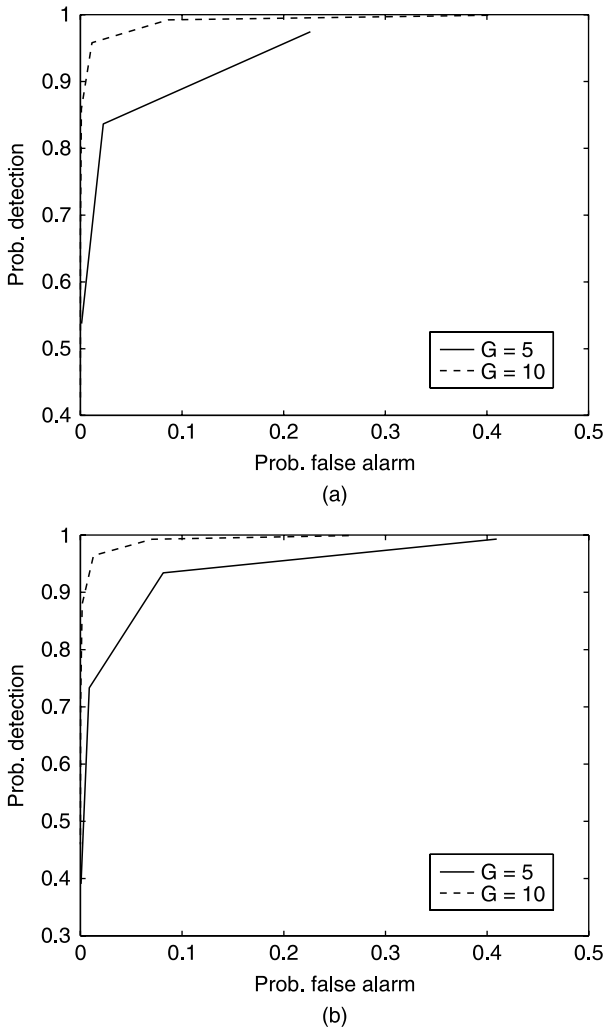
$$\text{PFA}(\gamma') = P(D_{\text{hard}} > \gamma'|H_0) = 1 - \sum_{k=0}^{\lfloor \gamma' G \rfloor} \binom{G}{k} p_0[1]^k (1 - p_0[1])^{G-k} \tag{7.24}$$

Thus, we see that the design of the hard decision fusion detector boils down to the choice of two thresholds: γ in Equation (7.18) that controls $p_1[1]$ and $p_0[1]$; and γ' in Equation (7.20) that, along with γ, controls the PFA and PD of the final detector. Since $E_i[\|\boldsymbol{Z}_k\|^2/N] = \sigma_i^2$ under H_i, the threshold γ can, in general, be chosen between σ_0^2 and $\sigma_1^2 = \sigma_0^2 + \sigma_s^2$ to yield a sufficiently low $p_0[1]$ (local PFA) and a corresponding $p_1[1] > p_0[1]$ (local PD). The threshold γ' can then be chosen between $p_0[1]$ and $p_1[1]$. To see this, note that the mean and variance of each U_k are: $E_i[U_k] = p_i[1]$ and $\text{var}_i[U_k] = p_i[1](1 - p_i[1])$, $i = 0, 1$. Again, by the law of large numbers, $E_i[L'] = p_i[1]$ and $\text{var}_i[L'] = p_i[1](1 - p_i[1])/G$, $i = 0, 1$. Thus, as long as $p_1[1] > p_0[1]$, which can be ensured via a proper choice of γ, the mean of l' is distinct under the two hypotheses and its variance goes to zero under both hypotheses as G increases. Let $\gamma' = p_0[1] + \delta$, where $0 < \delta < p_1[1] - p_0[1]$. Using Tchebyshev's inequality, as in soft decision fusion, it can be shown that for d_{hard}

$$\text{PD}(\gamma') = P(L' > \gamma'|H_1) \ge 1 - \frac{E_i[(L' - p_1[1])^2]}{(p_1[1] - \gamma')^2} = 1 - \frac{p_1[1](1 - p_1[1])}{G(p_1[1] - p_0[1] - \delta)^2} \tag{7.25}$$

$$\text{PFA}(\gamma') = P(L' > \gamma'|H_0) \le \frac{E[(L' - p_0[1])^2]}{\delta^2} = \frac{p_0[1](1 - p_0[1])}{G\delta^2} \tag{7.26}$$

© 2005 by Chapman & Hall/CRC



Figre 7.4. ROC curves for the hard decision fusion detector for different values of independent measurements G. The curve for each value of G is generated by varying γ' between $p_0[1]$ and $p_1[1]$. (a) $p_0[1] = 0.05$ and $p_1[1] = 0.52$; (b) $p_0[1] = 0.1$ and $p_1[1] = 0.63$.

Thus, as long as γ' is chosen to satisfy $p_0[1] < \gamma' < p_1[1]$, we attain perfect detector performance as $G \to \infty$.

Figure 7.4 plots the ROC curves for the energy detector based on hard decision fusion. The two chosen sets of values for $(p_0[1],\ p_1[1])$ are based on two different operating points on the $NG = 10$ ROC curve in Figure 7.3(a) for the soft decision fusion detector. Thus, the local hard decisions (u_k) corresponding to Figure 7.4 can be thought of as being based on $N = 10$-dimensional vectors. Then, the $G = 5$ curves for hard decision fusion in Figure 7.4 can be compared with the $NG = 50$ curve in Figure 7.3(a) for soft decision fusion. In soft decision fusion, the energies of the $N = 10$-dimensional vectors at the $G = 5$ independent nodes are combined to yield the $NG = 50$ curve in Figure 7.3(a). On the other hand, hard decisions based on $N = 10$-dimensional vectors at the $G = 5$ independent nodes are combined in hard decision fusion to yield the $G = 5$ curve in Figure 7.4(a). It is clear that the difference in performance between hard and soft decision fusion is significant. However, the $G = 10$ curve for hard decision fusion in Figure 7.4(a) yields better performance than the $NG = 50$ curve for soft decision

© 2005 by Chapman & Hall/CRC

fusion in Figure 7.3(a). Thus, hard decision fusion from ten SCRs performs better than soft decision fusion from five SCRs. Thus, we conclude that hard decision fusion from a sufficient number of SCRs may be more attractive (lower communication cost) than soft decision fusion. However, a more complete comparison requires carefully accounting for the communication cost of the two schemes.

7.4 Object Classification

In Section 7.3 we discussed object detection, deciding whether there is a vehicle present in a region R or not. The optimal detector compares the energy in the measurements with a threshold. Suppose the answer to the detection query is positive, i.e. there is a vehicle present in the region of interest. The next logical network query is to classify the vehicle. An example query is: Does the vehicle belong to class A, B, or C? Such a classification query is the focus of this section. We assume that the vehicle can be from one of M possible classes. Mathematically, this corresponds to choosing one out of M possible hypotheses, as opposed to two hypotheses in object detection.

In event (vehicle) detection, the performance was solely determined by signal energy in the two hypotheses. In single vehicle classification, we have to decide between M hypotheses. Thus, we need to exploit more detailed statistical characteristics (rather than energy) of the N-dimensional source signal vector $\boldsymbol{s}_k$ at each node. An imporant issue is what kind of N-dimensional measurements $\boldsymbol{z}_k$ should be collected at each node? This is the called *feature selection* [6]. Essentially, the raw time series data collected over the block time interval T_o at each node is processed to extract a relevant feature vector that best facilitates discrimination between different classes. Feature selection is a big research area in its own right, and we will not discuss it here; we refer the reader to Duda *et al.* [6] for a detailed discussion.

We will assume a particular type of feature vector — spectral feature vector — that can be obtained by computing a Fourier transform of the raw data [3]. This is a natural consequence of our signal model, in which the signals emitted by objects of interest are modeled as a stationary process [7]. Thus, we assume that the N-dimensional feature vector $\boldsymbol{z}_k$ is obtained by Fourier transformation of each block of raw time series data (whose length can be longer than N). An important consequence of Fourier features is that the different components of $\boldsymbol{s}_k$ correspond to different frequencies and are approximately statistically independent with the power in each component, $E[S_k^2[n]]$, proportional to a sample of the PSD associated with the vehicle class as defined in Equation (7.1); that is, $E[S_k^2[n]] \propto \Phi_s((n-1)/N)$, $n = 1, \ldots, N$. Furthermore, the statistics of $\boldsymbol{n}_k$ remain unchanged, since Fourier transformation does not change the statistics of white noise.

Mathematically, we can state the classification problem as an M-ary hypothesis testing problem

$$H_j\colon \boldsymbol{z}_k = \boldsymbol{s}_k + \boldsymbol{n}_k, \quad k = 1, \ldots, G, \quad j = 1, \ldots, M \tag{7.27}$$

where $\{\boldsymbol{n}_k\}$ are i.i.d. $\mathcal{N}(\boldsymbol{0}, \sigma_n^2 \boldsymbol{I}/n_G)$ as before but $\{\boldsymbol{s}_k\}$ are i.i.d. $\mathcal{N}(\boldsymbol{0}, \boldsymbol{\Lambda}_j)$ under H_j, where $\boldsymbol{\Lambda}_j$ is a diagonal matrix (since the different Fourier components of $\boldsymbol{s}_k$ are uncorrelated) with diagonal entries given by $\{\lambda_j[1], \ldots, \lambda_j[N]\}$, which are nonnegative and are proportional to samples of the PSD associated with class j as discussed above. Thus, under H_j, $\{\boldsymbol{z}_k\}$ are i.i.d. $\mathcal{N}(\boldsymbol{0}, \tilde{\boldsymbol{\Lambda}}_j)$, where $\tilde{\boldsymbol{\Lambda}}_j = \boldsymbol{\Lambda}_j + \sigma_n^2 \boldsymbol{I}/n_G$. Based on the measurement vectors $\{\boldsymbol{z}_k\}$ from the G SCRs, the manager node has to decide which one of the M classes the detected vehicle belongs to. We discuss CSP algorithms for classification based on fusion of both soft and hard decisions from each node.

7.4.1 Soft Decision Fusion

Assuming that different classes are equally likely, the optimal classifier chooses the class with the largest likelihood [5–7]:

$$C(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_G) = \arg \max_{j=1,\ldots,M} p_j(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_G) \tag{7.28}$$

© 2005 by Chapman & Hall/CRC

where $p_j(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_G)$ is the probability density function (pdf) of the measurements under H_j. Since the different measurements are i.i.d. zero-mean Gaussian, the joint pdf factors into marginal pdfs

$$p_j(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_G) = \prod_{k=1}^{G} p_j(\boldsymbol{z}_k) \tag{7.29}$$

$$p_j(\boldsymbol{z}_k) = \frac{1}{(2\pi)^{N/2}|\tilde{\boldsymbol{\Lambda}}_j|^{1/2}} \mathrm{e}^{-\frac{1}{2}\boldsymbol{z}_k^T \tilde{\boldsymbol{\Lambda}}_j^{-1} \boldsymbol{z}_k} \tag{7.30}$$

where $|\tilde{\boldsymbol{\Lambda}}_j| = \prod_{n=1}^{N}(\lambda_j[n] + \sigma_{\mathrm{n}}^2/n_G)$ denotes the determinant of $\tilde{\boldsymbol{\Lambda}}_j$ and $\boldsymbol{z}_k^T \tilde{\boldsymbol{\Lambda}}_j^{-1} \boldsymbol{z}_k = \sum_{n=1}^{N} z_k^2[n]/(\lambda_j[n] + \sigma_{\mathrm{n}}^2/n_G)$ is a weighted energy measure, where the weights depend on the vehicle class. It is often convenient to work with the negative log-likelihood functions

$$C(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_G) = \arg \min_{j=1,\ldots,M} l_j(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_G) \tag{7.31}$$

$$l_j(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_G) = -\frac{\log\ p_j(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_G)}{G} = -\frac{1}{G}\sum_{k=1}^{G} \log\ p_j(\boldsymbol{z}_k) \tag{7.32}$$

Note that the kth SCR has to communicate the log-likehood functions for all classes, $\log p_j(\boldsymbol{z}_k)$, $j = 1, \ldots, M$, based on the local measurement $\boldsymbol{z}_k$, to the manager node. Ignoring constants that do not depend on the class, the negative log-likelihood function for H_j takes the form

$$l_j(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_G) = \log|\tilde{\boldsymbol{\Lambda}}_j| + \frac{1}{G}\sum_{k=1}^{G} \boldsymbol{z}_k^T \tilde{\boldsymbol{\Lambda}}_j^{-1} \boldsymbol{z}_k \tag{7.33}$$

Thus, for each set of measurements $\{\boldsymbol{z}_k\}$ for a detected object, the classifier at the manager node computes l_j for $j = 1, \ldots, M$ and declares that the object (vehicle) belongs to the class with the smallest l_j.

A usual way of characterizing the classifier performance is to compute the average probability of error P_{e}, which is given by

$$P_{\mathrm{e}} = \frac{1}{M}\sum_{m=1}^{M} P_{\mathrm{e},m} \tag{7.34}$$

$$P_{\mathrm{e},m} = P(l_j < l_m \text{ for some } j \neq m\ |H_m) \tag{7.35}$$

where $P_{\mathrm{e},m}$ is the conditional error probability when the true class of the vehicle is m. Computing $P_{\mathrm{e},m}$ is complicated, in general, but we can bound it using the union bound [5]

$$P_{\mathrm{e},m} \leq \sum_{j=1,\, j\neq m}^{M} P(l_j < l_m | H_m) \tag{7.36}$$

Note that $P_{\mathrm{e}} = 1 - \mathrm{PD}$, where PD denotes the average probability of correct classification:

$$\mathrm{PD} = \frac{1}{M}\sum_{m=1}^{M} \mathrm{PD}_m \tag{7.37}$$

$$\mathrm{PD}_m = P(l_m \leq l_j \text{ for all } j \neq m | H_m) \tag{7.38}$$

© 2005 by Chapman & Hall/CRC

and PD_m denotes the probability of correct classification conditioned on H_m. The pairwise error probabilities on the right-hand side of Equation (7.36) can be computed analytically, but they take on complicated expressions [5,7]. However, it is relatively easy to show that, as the number of independent measurements G increases, P_e decreases and approaches zero (perfect classification) in the limit. To see this, note from Equation (7.32) that, by the law of large numbers [5], under H_m we have

$$\lim_{G\to\infty} l_j(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_G) = -E_m[\log\, p_j(\boldsymbol{Z})] = D(p_m \| p_j) + h_m(\boldsymbol{Z}) \tag{7.39}$$

where $D(p_m\|p_j)$ is the Kullback–Leibler (K–L) distance between the pdfs p_j and p_m and $h_m(\boldsymbol{Z})$ is the differential entropy of $\boldsymbol{Z}$ under H_m [8]:

$$D(p_m\|p_j) = E_m\big[\log(p_m(\boldsymbol{Z})/p_j(\boldsymbol{Z}))\big] = \log\Big(|\tilde{\boldsymbol{\Lambda}}_j|/|\tilde{\boldsymbol{\Lambda}}_m|\Big) + \mathrm{tr}\Big(\tilde{\boldsymbol{\Lambda}}_j^{-1}\tilde{\boldsymbol{\Lambda}}_m - \boldsymbol{I}\Big) \tag{7.40}$$

$$h_m(\boldsymbol{Z}) = -E_m[\log p_m(\boldsymbol{Z})] = \frac{1}{2}\log\Big((2\pi\mathrm{e})^N|\tilde{\boldsymbol{\Lambda}}_m|\Big) \tag{7.41}$$

Note that $\mathrm{tr}(\cdot)$ denotes the trace of a matrix (sum of the diagonal entries). An important property of the K–L distance is that $D(p_m\|p_j) > 0$ unless $p_m = p_j$, i.e. the densities for class j and m are identical (in which case there is no way to distinguish between the two classes). Thus, from Equation (7.39) we conclude that, under H_m, l_m will always give the smallest value and thus lead to the correct decision as $G \to \infty$ as long as $D(p_j\|p_m) > 0$ for all $j \neq m$. For more discussion on performance analysis of soft decision fusion, we refer the reader to D'Costa and Sayeed [7].

7.4.2 Hard Decision Fusion

In soft decision fusion, the kth SCR sends M log-likelihood values $\{\boldsymbol{z}_k^T\tilde{\boldsymbol{\Lambda}}_j^{-1}\boldsymbol{z}_k :\ j = 1, \ldots, M\}$ for the M classes, computed from its local measurement vector $\boldsymbol{z}_k$, to the manager node. All these local likelihood values are real-valued and thus require many bits for accurate and reliable digital communication. The number of bits required for accurate communication can be estimated from the differential entropy of the likelihoods [7,8]. While exchange of real-valued likelihoods puts much less communication burden on the network compared with data fusion in which the feature vectors $\{\boldsymbol{z}_k\}$ are communicated from each SCR to the manager node, it is attractive to reduce the communication burden even further. One way is to quantize the M likelihood values from different SCRs with a sufficient number of bits. Another natural quantization strategy is to compute local hard decisions in each SCR based on the local measurement vector $\boldsymbol{z}_k$, analogous to the approach in object detection. In this section we discuss this hard decision fusion approach.

We assume that in the kth SCR a hard decision is made about the object class based on the local measurement vector $\boldsymbol{z}_k$:

$$u_k(\boldsymbol{z}_k) = \arg\max_{j=1,\ldots,M} p_j(\boldsymbol{z}_k), \quad k = 1, \ldots, G \tag{7.42}$$

Equivalently, the decision could be made based on the negative log-likelihood function. Note that u_k maps $\boldsymbol{z}_k$ to an element in the set of classes $\{1, \ldots, M\}$ and is thus a discrete random variable with M possible values. Furthermore, since all $\{\boldsymbol{z}_k\}$ are i.i.d., so are $\{u_k\}$. Thus, the hard decision random variable U (we ignore the subscript k) is characterized by a probability mass function (pmf) under each hypothesis. Let $\{p_m[j]\colon j = 1, \ldots, M\}$ denote the M values of the pmf under H_m. The pmfs for all hypotheses are described by the following probabilities:

$$p_m[j] = P(U(\boldsymbol{z}_k) = j|H_m) = P(p_j(\boldsymbol{z}_k) > p_l(\boldsymbol{z}_k) \text{ for all } l \neq j\ |H_m), \quad j, m = 1, \ldots, M \tag{7.43}$$

© 2005 by Chapman & Hall/CRC

The hard decisions $\{u_k\}$ from all SCRs are communicated to the manager node, which makes the final decision as

$$C_{\text{hard}}(u_1, \ldots, u_G) = \arg \max_{j=1,\ldots,M} p_j[u_1, \ldots, u_G] \tag{7.44}$$

where

$$p_j[u_1, \ldots, u_G] = \prod_{k=1}^{G} p_j[u_k] \tag{7.45}$$

since the $\{u_k\}$'s are i.i.d. Again, we can write the classifier in terms of negative log-likelihoods:

$$C_{\text{hard}}(u_1, \ldots, u_G) = \arg \min_{j=1,\ldots,M} l'_j[u_1, \ldots, u_G] \tag{7.46}$$

$$l'_j[u_1, \ldots, u_G] = -\frac{1}{G} \log p_j[u_1, \ldots, u_G] = -\frac{1}{G} \sum_{k=1}^{G} \log p_j[u_k] \tag{7.47}$$

and while the exact calculation of the probability of error is complicated, it can be bounded via pairwise error probabilities analogous to the soft decision classifier. Similarly, we can say something about the asymptotic performance of the hard decision classifier as $G \to \infty$. Note from Equation (7.47) that, because of the law of large numbers, under H_m we have

$$\lim_{G\to\infty} l'_j[u_1, \ldots, u_G] = -E_m[\log\, p_j[U]] = D(p_m \| p_j) + H_m(U) \tag{7.48}$$

where $D(p_m \| p_j)$ is the K–L distance between the pmfs p_m and p_j and $H_m(U)$ is the entropy of the hard decision under H_m [8]:

$$D(p_m \| p_j) = \sum_{i=1}^{M} p_m[i] \log(p_m[i]/p_j[i]) \tag{7.49}$$

$$H_m(U) = -\sum_{i=1}^{M} p_m[i] \log p_m[i] \tag{7.50}$$

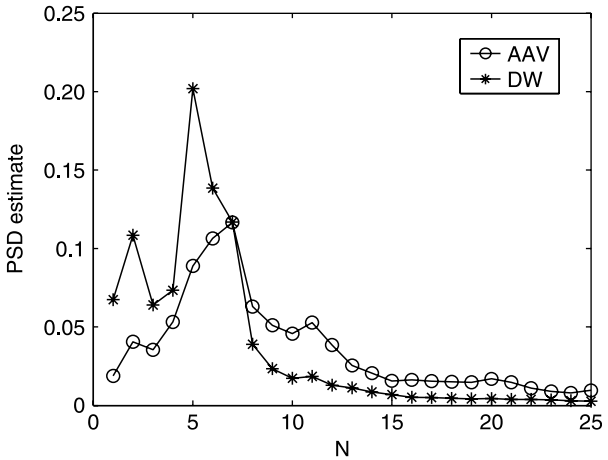
Thus, we see from Equation (7.48) that, in the limit of large G, we will attain perfect classification performance as long as $D(p_m \| p_j) > 0$ for all $j \neq m$.

7.4.3 Numerical Results

We present some numerical results to illustrate soft decision classification.[3] We consider classification of a single vehicle from $M = 2$ possible classes: Amphibious Assault Vehicle (AAV; tracked vehicle) and Dragon Wagon (DW; wheeled vehicle). We simulated $N = 25$-dimensional (averaged) acoustic Fourier feature vectors for $K = Gn_G = 10$ nodes in G SCRs (n_G nodes in each SCR) for different values of G and n_G. The diagonal matrices $\mathbf{\Lambda}_1$ (AAV) and $\mathbf{\Lambda}_2$ (DW) corresponding to PSD samples were estimated from

[3]The results are based on real data collected as part of the DARPA SensIT program.

© 2005 by Chapman & Hall/CRC



Figre 7.5. Covariance matrix eigenvalues (PSD estimates) for AAV and DW.

measured experimental data. The PSD estimates are plotted in Figure 7.5 for the two vehicles. In addition to the optimal soft decision fusion classifier C, two sub-optimal classifiers were also simulated: (i) a decision-fusion classifier C_{df} that assumes that all measurements are independent (optimal for $K = G$); (ii) a data-averaging classifier C_{da} that treats all measurements as perfectly correlated (optimal for $K = n_G$). For each H_j, the G statistically independent source signal vectors $\boldsymbol{s}_k$ were generated using $\boldsymbol{\Lambda}_j$ as

$$\boldsymbol{s}_k = \boldsymbol{\Lambda}_j^{1/2}\mathbf{v}_k, \quad k = 1, \ldots, G \tag{7.51}$$

where $\mathbf{v}_k \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. Then, the n_G noisy measurements for the kth SCR were generated as

$$\boldsymbol{z}_{k,i} = \boldsymbol{s}_k + \boldsymbol{n}_{k,i}, \quad i = 1, \ldots, n_G, \quad k = 1, \ldots, G \tag{7.52}$$

where $\boldsymbol{n}_{k,i} \sim \mathcal{N}(\mathbf{0}, \sigma_n^2\boldsymbol{I})$. The average probability of correct classification, $\text{PD} = 1 - P_{\text{e}}$, for the three classifiers was estimated using Monte Carlo simulation over 5000 independent trials.

Figure 7.6 plots PD as a function of the SNR for the three classifiers for $K = 10$ and different combinations of G and n_G. As expected, C and C_{da} perform identically for $K = n_G$ (perfectly correlated measurements); see Figure 7.6(a). On the other hand, C and C_{df} perform identically for $K = G$ (perfectly independent measurements); see Figure 7.6(d)). Note that C_{df} incurs a small loss in performance compared with C in the perfectly correlated (worst) case, which diminishes at high SNRs. The performance loss in C_{da} in the independent (worst) case is very significant and does not improve with SNR.[4] Thus, we conclude that the sub-optimal decision-fusion classifier C_{df} that ignores correlation in the measurements (and thus avoids the high-bandwidth data fusion of feature vectors in each SCR for signal averaging) closely approximates the optimal classifier, except for an SNR loss. It can be shown that the SNR loss is proportional to n_G, since C_{df} does not perform signal averaging in each SCR for noise reduction [7]. Furthermore, it can also be shown that C_{df} yields perfect classification performance (just as the optimal classifier) as $G \to \infty$ under mild conditions on the signal statistics, analogous to those for the optimal classifier [7]. Thus, the sub-optimal decision-fusion classifier (with either hard or soft decisions) is a very attractive choice in sensor

[4]It can be shown that, at high SNR, all events are classified as DW by C_{da}, since $\log|\Lambda_{\text{DW}}| < \log|\Lambda_{\text{AAV}}|$ due to the peakier eigenvalue distribution for DW [7], as evident from Figure 7.5.

© 2005 by Chapman & Hall/CRC

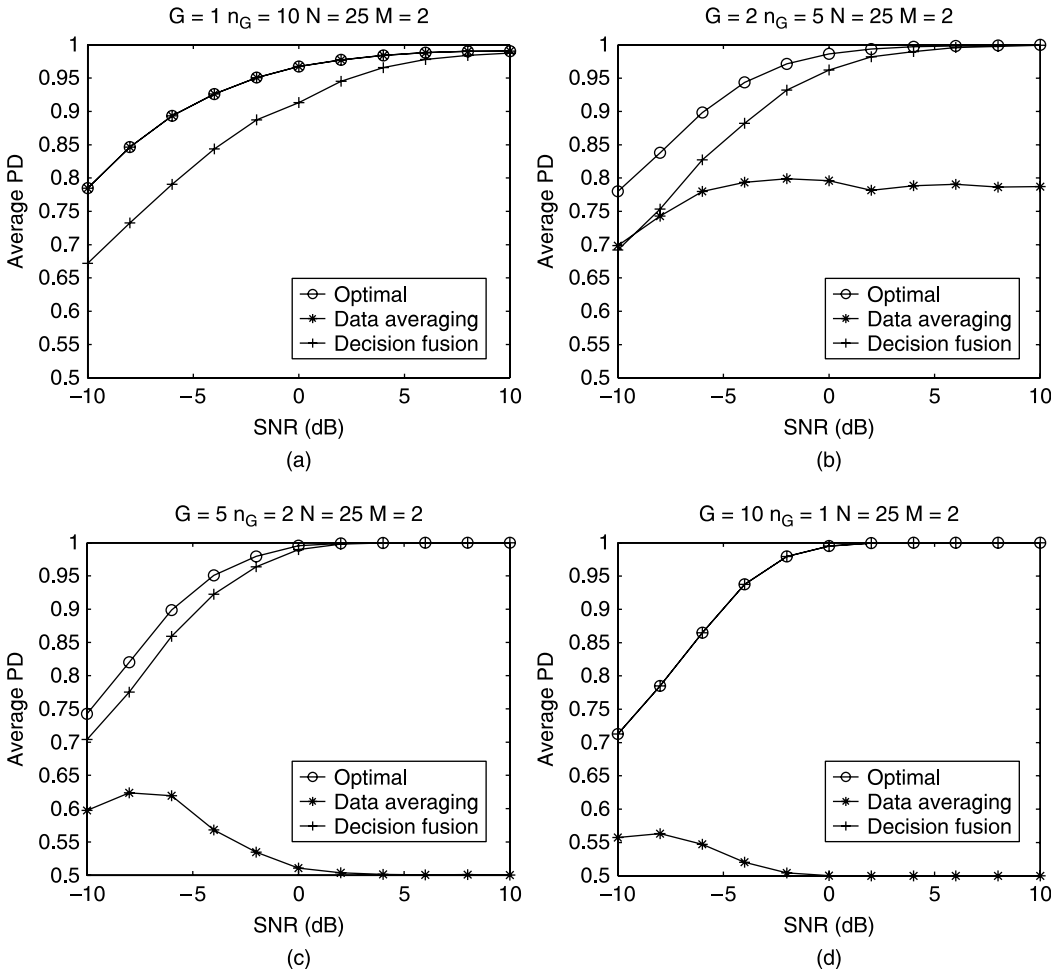


Figure 7.6. PD of the three classifiers versus SNR. (a) $K = n_G = 10$ (perfectly correlated measurements). (b) $G = 2$ and $n_G = 5$. (c) $G = 5$ and $n_G = 2$. (d) $K = G = 10$ (independent measurements).

networks because it puts the least communication burden on the network (avoids data fusion in each SCR).

7.5 Conclusions

Virtually all applications of sensor networks are built upon two primary operations: (i) distributed processing of data collected by the nodes; (ii) communication and routing of processed data from one part of the network to another. Furthermore, the second operation is intimately tied to the first operation, since the information flow in a sensor network depends directly on the data collected by the nodes. Thus, distributed signal processing techniques need to be developed in the context of communication and routing algorithms and *vice versa*.

In this chapter we have discussed distributed decision making in a simple context — detection and classification of a single object — to illustrate some basic principles that govern the interaction between information processing and information routing in sensor networks. Our approach was based on modeling the object signal as a band-limited random field in space and time. This simple model partitions the network into disjoint SCRs whose size is inversely proportional to the spatial signal

© 2005 by Chapman & Hall/CRC

bandwidths. This partitioning of network nodes into SCRs suggests a structure on information exchange between nodes that is naturally suited to the communication constraints in the network: high-bandwidth feature-level data fusion is limited to spatially local nodes within each SCR, whereas global fusion of low-bandwidth local SCR decisions is sufficient at the manager node. We showed that data averaging within each SCR improves the effective measurement SNR, whereas decision-fusion across SCRs combats the inherent statistical variability in the signal. Furthermore, we achieve perfect classification in the limit of large number of SCRs (large number of independent measurements). This simple structure on the nature of information exchange between nodes applies to virtually all CSP algorithms, including distributed estimation and compression.

Our investigation based on the simple model suggests several interesting directions for future studies.

7.5.1 Realistic Modeling of Communication Links

We assumed an ideal noise-free communication link between nodes. In practice, the communication link will introduce some errors which must be taken into account to obtain more accurate performance estimates. In the context of detection, there is considerable work that can be made to bear on this problem [9]. Furthermore, the object signal strength sensed by a node will depend on the distance between the node and the object. This effect should also be included in a more detailed analysis. Essentially, this will limit the size of the region over which node measurements can be combined — the nodes beyond a certain range will exhibit very poor measurement SNR.

7.5.2 Multi-Object Classification

Simultaneous classification of multiple objects is a much more challenging problem. For example, the number of possible hypotheses increases exponentially with the number of objects. Thus, simpler distributed classification techniques are needed. Several forms of sub-optimal algorithms, including tree-structured classifiers [6] and subspace-based approaches [10,11], could be exploited in this context. Furthermore, we have only discussed some particular forms of soft and hard decision fusion in this chapter. There are many (sub-optimal) possibilities in general [12] which could be explored to best suit the needs of a particular application.

7.5.3 Nonideal Practical Settings

We have investigated distributed decision making under idealized assumptions to underscore some basic underlying principles. The assumptions are often violated in practice and must be taken into account to develop robust algorithms [3]. Examples of nonideality include nonstationary signal statistics (which may arise due to motion or gear-shifts in a vehicle), variability in operating conditions compared with those encountered during training, and faulty sensors. Training of classifiers, which essentially amounts to estimating object statistics, is also a challenging problem [6]. Finally, Gaussian modeling of object statistics may not be adequate; non-Gaussian models may be necessary.

References

[1] Estrin, D. *et al.*, Instrumenting the world with wireless sensor networks, in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing 2001*, vol. 4, 2033, 2001.

[2] Kumar, S. *et al.* (eds), Special issue on collaborative signal and information processing in microsensor networks, *IEEE Signal Processing Magazine*, (March), 2002.

[3] Li, D. *et al.*, Detection, classification, tracking of targets in microsensor networks, *IEEE Signal Processing Magazine*, (March), 17, 2002.

[4] Stark, H. and Woods, J.W., *Probability, Random Processes, and Estimation Theory for Engineers*, Prentice Hall, New Jersey, 1986.

[5] Proakis, J.G., *Digitial Communications*, 3rd ed., McGraw Hill, New York, 1995.

© 2005 by Chapman & Hall/CRC

[6] Duda, R. *et al.*, *Pattern Classification*, 2nd ed., Wiley, 2001.
[7] D'Costa, A. and Sayeed, A.M., Collaborative signal processing for distributed classification in sensor networks, in *Lecture Notes in Computer Science (Proceedings of IPSN'03)*, Zhao, F. and Guibas, L. (eds), Springer-Verlag, Berlin, 193, 2003.
[8] Cover, T.M. and Thomas, J.A., *Elements of Information Theory*, Wiley, 1991.
[9] Varshney, P.K., *Distributed Detection and Data Fusion*, Springer, 1996.
[10] Fukunaga, K. and Koontz, W.L.G., Application of the Karhunen–Loeve expansion to feature selection and ordering, *IEEE Transactions on Computers*, C-19, 311, 1970.
[11] Watanabe, S. and Pakvasa, N., Subspace method to pattern recognition, in *Proceedings of the 1st International Conference on Pattern Recognition*, 25, February 1973.
[12] Kittler, J. *et al.*, Advances in statistical feature selection, *Advances in Pattern Recognition — ICAPR 2001, Second International Conference Rio de Janeiro, Proceedings*, vol. 2013, 425, March 2001. ICAPR 2001 — Electronic Edition (Springer LINK).

© 2005 by Chapman & Hall/CRC

8

Parameter Estimation

David S. Friedlander

8.1 Introduction

Most of the work presented in this chapter was done under two research projects: *Semantic Information Fusion* and *Reactive Sensor Networks*. These projects were performed at the Penn State University Applied Research Laboratory and funded under DARPA's Sensor Information Technology program (see Section 8.8). Experimental results were obtained from field tests performed jointly by the program participants.

Parameters measured by sensor networks usually fall into three categories: environmental parameters such as wind speed, temperature, or the presence of some chemical agent; target features used for classification; and estimates of position and velocity along target trajectories. The environmental parameters are generally local. Each measurement is associated with a point in space (the location of the sensor) and time (when the measurement was taken). These can be handled straightforwardly by sending them to the data sink via whatever networking protocol is being used. Other parameters may need to be determined by multiple sensor measurements integrated over a region of the network. Techniques for doing this are presented in this chapter.

These parameters are estimated by combining observations from sensor platforms distributed over the network. It is important for the network to be robust in the sense that the loss of any given platform or small number of platforms should not destroy its ability to function. We may not know ahead of time exactly where each sensor will be deployed, although its location can be determined by a global positioning system after deployment. For these reasons, it is necessary for the network to self-organize [1]. In order to reduce power consumption and delays associated with transmitting large amounts of information over long distances, we have designed an algorithm for dynamically organizing platforms into clusters along target trajectories.

This algorithm is based on the concept of space–time neighborhoods. The platforms in each neighborhood exchange information to determine target parameters, allowing multiple targets to be processed in parallel and distributed power requirements over multiple platforms. A neighborhood N is a set of space–time points defined by

$$N \equiv \left\{(\vec{x}', t'): \left|\vec{x} - \vec{x}'\right| < \Delta x \quad \text{and} \quad \left|t - t'\right| < \Delta t\right\}$$

© 2005 by Chapman & Hall/CRC

where Δx and Δt define the size of the neighborhood in space and time. A dynamic space–time window $w(t)$ around a moving target with trajectory $\bar{g}(t)$ is defined by

$$w(t) \equiv \left\{(\bar{x}', t): \left|\bar{g}(t) - \bar{x}'\right| < \Delta x \quad \text{and} \quad \left|t' - t\right| < \Delta t\right\}$$

We want to solve for $\bar{g}(t)$ based on sensor readings in the dynamic window $w(t)$. Most sensor readings will reach a peak at the closest point of approach (CPA) of the target to the sensor platform. We call these occurrences CPA events. In order to filter out noise and reflections, we count only peaks above a set threshold and do not allow more than one CPA event from a given platform within a given dynamic window.

Assuming we know the locations of each platform and that each platform has a reasonably accurate clock, we can assign a space–time point to each CPA event. We define the platforms with CPA events within a given dynamic window as a *cluster*. Platforms within a given cluster exchange information to define target parameters within the associated space and time boundaries. This technique can be easily extended to include moving platforms, as long as the platform trajectories are known and their velocities are small compared with the propagation speed of the energy field measured by the sensors. Typically, this would be the speed of light or the speed of mechanical vibrations, such as sound.

8.2 Self-Organization of the Network

We now show how to determine platform clusters along target trajectories [2]. The clusters are defined by dynamic space–time windows of size Δx by Δt. Ideally, the window boundaries would also be dynamic. For example, we want Δx to be large compared with the platform density and small compared with the target density, and we want $\Delta x \approx v_t \Delta t$, where v_t is a rough estimate of the target velocity, possibly using the previously calculated value. In practice, we have obtained good results with constant values for Δx and Δt in experiments where the target density was low and the range of target velocities was not too large.

The algorithm for determining clusters is shown in Figure 8.1. Each platform contains two buffers, one for the CPA events it has detected and another for the events detected by its neighbors. The CPA Detector looks for CPA events. When it finds one, it stores the amplitude of the peak, time of the peak, and position of the platform in a buffer and broadcasts the same information to its neighbors. When it receives neighboring CPA events, it stores them in another buffer. The *Form Clusters* routine looks at each CPA event in the local buffer. A space–time window is determined around each local event. All of the neighboring events within the window are compared with the local event. If the peak amplitude of

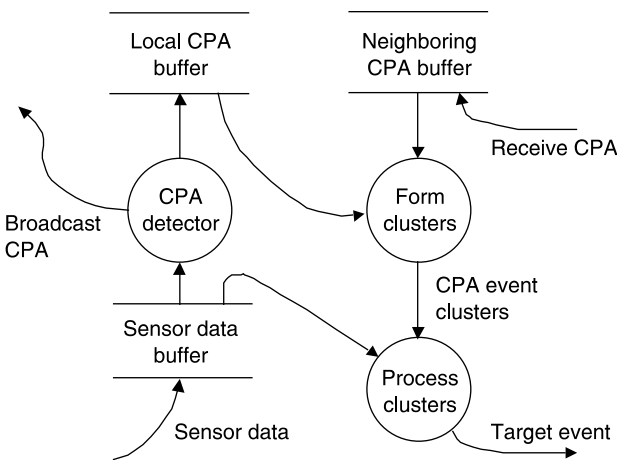


Figure 8.1. Cluster formation process.

© 2005 by Chapman & Hall/CRC

```
For each local CPA event k_ij = k(x_i, t_j)
    For each neighboring CPA event n_kl = n(x_l, t_k)
        If n_kl is in the neighborhood N_ij = N(x̄_i, t_j)
            Add n_kl to the event set M
    If the local peak amplitude a(k_ij) ≥ a(n_kl) ∀ n_kl ∈ M
        Process CPA event cluster F ≡ k_ij ∪ M
```

Figure 8.2. *Form Clusters* pseudo-code.

the local event is greater than that of its neighbors within the window, then the local platform elects itself as the *cluster head*. The cluster head processes its own and its neighbor's relevant information to determine target parameters. If a platform has determined that it is not the cluster head for a given local event, then the event is not processed by that platform. If the size of the window is reasonable, then the method results in efficient use of the platforms and good coverage of the target track. Pseudo-code for this process is shown in Figure 8.2. The *Process Clusters* routine then determines the target position, velocity, and attributes as described below.

8.3 Velocity and Position Estimation

8.3.1 Dynamic Space–Time Clustering

We have extended techniques found in Hellebrant *et al.* [3] for velocity and position estimation [2]. We call the method *dynamic space–time clustering* [4]. The example shown below is for time and two spatial dimensions, $\bar{x} = (x, y)$; however, its extension to three spatial dimensions is straightforward. The technique is a parameterized linear regression. The node selected as the cluster head, n_0, located at position (x_0, y_0), estimates velocity and position. We estimate the target location and velocity at time t_0, the time of CPA for node n_0. This node has information and observations from a set of other nodes in a cluster around it. Denote the cluster around n_0 as $F \equiv \{n_i | \bar{x}_i - \bar{x}_0| < \Delta x \text{ and } |t_0 - t_i| < \Delta t\}$, where Δx and Δt are bounds in space and time. We defined the spatial extent of the neighborhoods so that vehicle velocities are approximately linear [3].

The position of a given node n_i in the cluster is $\bar{x}_i$ and the time of its CPA is t_i. This forms a space–time sphere around the position (x_0, y_0, t_0). The data are divided into one set for each special dimension; in our case: $(t_0, x_0), (t_1, x_1), \ldots, (t_n, x_n)$ and $(t_0, y_0), (t_1, y_1), \ldots, (t_n, y_n)$. We then weighted the observations based on the CPA peak amplitudes, on the assumption that CPA times are more accurate when the target passes closer to the sensor, to give $(x_0, t_0, w_0,), (x_1, t_1, w_1,), \ldots (x_n, t_n, w_n,)$ and $(y_0, t_0, w_0,), (y_1, t_1, w_1,), \ldots, (y_n, t_n, w_n)$, where w_i is the weight of the ith event in the cluster. This greatly improved the quality of the predicted velocities. Under these assumptions, we can apply least-squares linear regression to obtain the equations $x(t) = v_x t + c_1$ and $y(t) = v_y t + c_2$, where:

$$v_x = \frac{\sum_i t_i \sum_i x_i - \sum_i w_i \sum_i x_i t_i}{\left(\sum_i t_i\right)^2 - \sum_i w_i \sum_i t_i^2}, \quad v_y = \frac{\sum_i t_i \sum_i y_i - \sum_i w_i \sum_i y_i t_i}{\left(\sum_i t_i\right)^2 - \sum_i w_i \sum_i t_i^2}$$

and the position $\bar{x}(t_0) = (c_1, c_2)$. The space–time coordinates of the target for this event are $(\bar{x}(t_0), t_0)$.

This simple technique can be augmented to ensure that changes in the vehicle trajectory do not degrade the quality of the estimated track. The correlation coefficients for the velocities in each spatial dimension (r_x, r_y) can be used to identify large changes in vehicle direction and thus limit the CPA event

© 2005 by Chapman & Hall/CRC

```
Input: Time-sorted event cluster F of CPA values.
Output: Estimated velocity components v_x and v_y.

While |F| ≥ 5 {
    Compute v_x and v_y using event cluster F;
    Compute r_x and r_y ; the v_x and v_y velocity
                        ; correlation coefficients for F
    If r_x > R_x || r_y > R_y
    {
        R_x = r_x;
        R_y = r_y
        v_x_store = v_x;
        v_y_store = v_y;
    }
    PopBack(F);
};
```

Figure 8.3. Velocity calculation algorithm.

cluster to include only those nodes that will best estimate local velocity. Assume that the observations are sorted as follows: $o_i < o_j \Rightarrow |t_i - t_0| < |t_j - t_0|$, where o_t is an observation containing a time, location, and weight. The velocity elements are computed once with the entire event set. After this, the final elements of the list are removed and the velocity is recomputed. This process is repeated while at least five CPAs are present in the set; subsequently, the event subset with the highest velocity correlation is used to determine velocity. Estimates using fewer than five CPA points can bias the computed velocity and reduce the accuracy of our approximation. Figure 8.3 summarizes our technique.

Once a set of position and velocity estimates has been obtained, they are integrated into a track. The tracking algorithms improve the results by considering multiple estimates [5–7].

Beamforming is another method for determining target velocities [8]. Beamforming tends to be somewhat more accurate than dynamic space–time clustering, but it uses much greater resources. A comparison of the two methods is given in Phoha *et al.* [4].

8.3.2 Experimental Results for Targets Velocities

We have analyzed our velocity estimation algorithm using the field data these results appear in Table 8.1. Figures 8.4 and 8.5 show plots displaying the velocity estimations.

8.4 Moving Target Resolution

We developed new results on estimating the capacity of a sensor network to handle moving targets. Theoretically, target velocity can be determined from three platforms. Our analysis of the data shows that five are necessary for good accuracy and stability; see Figure 8.6.

Table 8.1. Quality of estimation

Computed vs. True Velocity	Percent
Percent within 1 m/s	81%
Percent within 2 m/s	91%
Percent within 5°	64%
Percent within 11°	80%
Percent within 17°	86%

© 2005 by Chapman & Hall/CRC

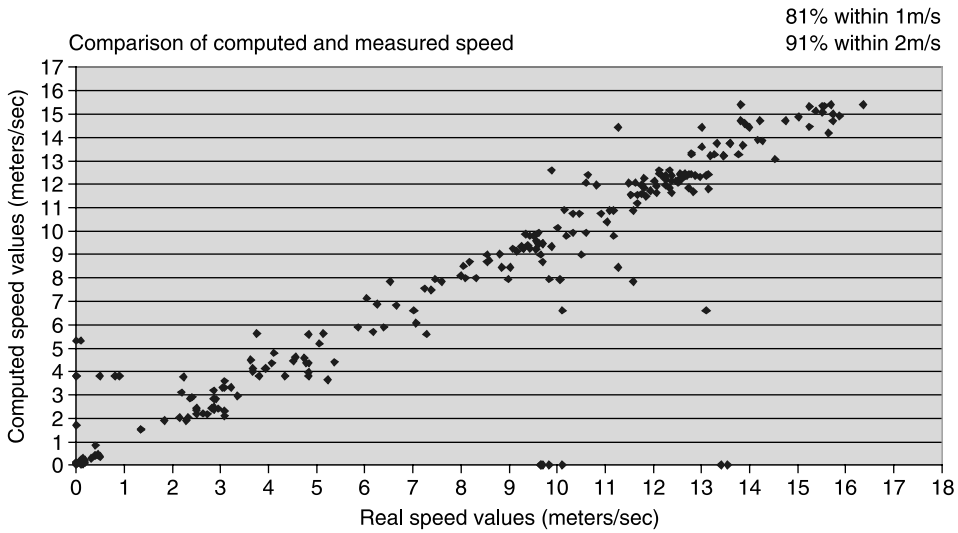


Figure 8.4. Computed speed vs. true speed (field test).

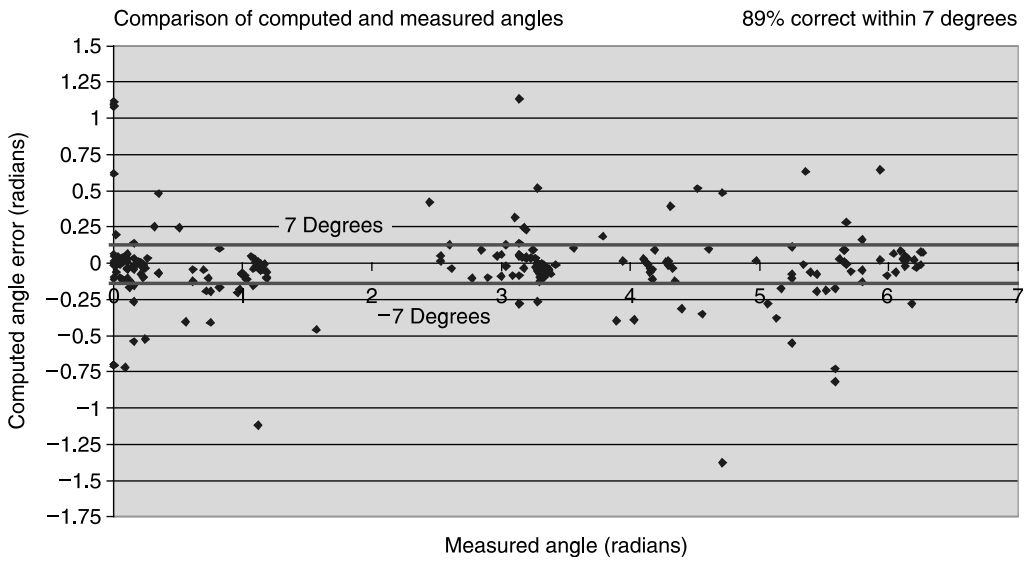


Figure 8.5. Computed angle vs. true angle (field test).

As shown in Figure 8.6, the radius of the spatial window for resolving a target's velocity is $r \approx \sqrt{5/\pi\rho_p}$ where r is the radius and ρ_p is the platform density. This gives us approximately five nodes in a space–time window, as required in Section 8.3. The amount of time needed to collect these data is determined by the time it takes the target to cross the spatial window Δt and the network latency δ: $\Delta t \approx (2r/v) + \delta$, where v is the target velocity; i.e. platforms in the window can be separated by a distance of up to $2r$. Two given target trajectories can be resolved unless $\exists t, t'$: $|\bar{x}_1(t) - \bar{x}_2(t)| \leq 2r$ and $|t - t'| \leq \Delta t$, where $\bar{x}_i(t)$ is the trajectory of target i.

We can define the target density ρ_t as the density of targets in a reference frame moving with the target velocity v_t or, equivalently, the density of targets in a "snapshot" of the moving targets. We can

© 2005 by Chapman & Hall/CRC

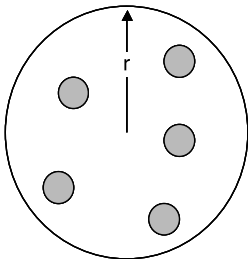


Figure 8.6. Sensor network area needed to determine target velocity.

have only one target at a time in the area shown in Figure 8.6, so $\rho_t \leq \rho_p/5$. The maximum capacity of the network in targets per second per meter of perimeter is given by $J_{max} \approx \rho_p v_t/5$. This is based on the assumption that the acoustic signals from two targets spaces approximately $2r = 2\sqrt{5/\pi\rho_p}$ meters apart will not interfere to the point where their peaks cannot be distinguished.

8.5 Target Classification Using Semantic Information Fusion

The semantic information fusion (SIF) technique described in this section was developed and applied to acoustic data [1]. It should be applicable to other scalar data, such as seismic sensors, but may not apply to higher dimensional data, such as radar. The method identifies the presence or absence of target features (attributes) detectable by one or more sensor types. Its innovation is to create a separate database for each attribute–value pair under investigation. Since it uses principal component analysis (PCA) [9], data from different types of sensor can be integrated in a natural way. The features can be transmitted directly or used to classify the target. PCA uses singular value decomposition (SVD), a matrix decomposition technique that can be used to reduce the dimension of time series data and improve pattern-matching results.

SIF processing consists of offline and online stages. The offline processing is computationally intensive and includes SVD of vectors whose components are derived from the time series of multiple channels. The attributes are expressed as mutually exclusive alternatives such as *wheeled* (or *tracked*), *heavy* (or *light*), *diesel engine* (or *piston engine*), etc.

Typically, the time series are transformed by a functional decomposition technique such as Fourier analysis. More recently developed methods, such as those by Goodwin and Vaidyanathan are promising. The spectral vectors are merged with the attribute data to from the pattern-matching database. The database vectors are then merged into a single matrix M, where each column of the matrix is one of the vectors. The order of the columns does not matter. The matrix is then transformed using SVD. The results include a set of reduced-dimension pattern-matching exemplars that are preloaded into the sensor platforms.

SVD produces a square matrix Σ and rectangular, unitary matrices U and V such that $M = U\Sigma V^T$. The dimension of M is $m \times n$, where n is the number of vectors and m is the dimension of each vector (the number of spectral components plus attribute dimensions). The dimension of Σ is $k \times k$, where k is the rank of M, where Σ is a diagonal matrix containing the eignvalues of M in decreasing order. The dimension of U is $m \times k$ and the dimension of V^T is $k \times n$.

The number of significant eignvalues r is then determined. The matrix Σ is truncated to a square matrix containing only the rth largest eignvalues. The matrix U is truncated to be $m \times r$ and V^T to be $r \times n$. This results in a modified decomposition: $M \approx \hat{U}\hat{\Sigma}\hat{V}^T$, where $\hat{U}$ is an $m \times r$ matrix containing the first k columns of U, $\hat{\Sigma}$ is an $r \times r$ matrix containing the first r rows and columns of Σ, and $\hat{V}^T$ is $r \times n$ matrix containing the first r rows of V^T.

The online, real-time processing is relatively light. It consists of taking the power spectrum of the unknown time series data; forming the unknown sample vector; a matrix multiplication to convert the

© 2005 by Chapman & Hall/CRC

unknown sample vector into the reduced dimensional space of the pattern database; and vector dot products to determine the closest matches in the pattern database. Since the pattern matching is done on all of the peaks in the event neighborhood, an estimate of the uncertainty in the target attributes can also be calculated.

The columns of $\hat{V}^{\mathrm{T}}$ comprise the database for matching against the unknown reduced-dimensional target vector. If we define the ith column of $\hat{V}^{\mathrm{T}}$ as $\hat{p}^i$, the corresponding column of M as $\bar{p}^i$, the unknown full-dimensional target vector as $\bar{q}$, and $\hat{q} = (\bar{q})^{\mathrm{T}}\hat{U}\hat{\Sigma}^{-1}$ as the reduced-dimensional target vector, then the value of the match between $\hat{p}^i$ and the target is $\hat{p}^i \cdot \hat{q}$. We can define the closest vector to $\hat{q}$ as $\hat{p}^m$ where $m = index \max_i(\hat{p}^i \cdot \hat{q})$. We then assign the attribute values of $\bar{p}^m$, the corresponding full dimensional vector, to the unknown target. The results might be improved using a weighted sum, say $w_i \sim 1/|\hat{q} - \hat{p}^m|$, of the attribute values (zero or one) of the k closest matches as the target attributes' values, i.e. $\bar{q}' = \sum_{i=1}^{k} w_i \bar{p}^i$. This would result in attributes with a value between zero and one instead of zero or one, which could be interpreted as the probability or certainty that the target has a given attribute.

Two operators, which are trivial to implement algorithmically, are defined for the SIF algorithms. If M is an $(r \times c)$ matrix, and $\bar{x}$ is a vector of dimension r, then

$$M \oplus \bar{x} \equiv \begin{bmatrix} m_{11} & \cdots & m_{1c} & x_1 \\ \vdots & \ddots & \vdots & \vdots \\ m_{r1} & \cdots & m_{rc} & x_r \end{bmatrix}$$

If $\bar{x}$ is a vector of dimension n and $\bar{y}$ is a vector of dimension m, then $\bar{x} \,||\, \bar{y} \equiv (x_1, x_2, \ldots, x_n, y_1, y_2, \ldots y_m)$. The offline algorithm for SIF is shown in Figure 8.7.

The matrices $\hat{U}$, $\hat{\Sigma}$, and $\hat{V}^{\mathrm{T}}$ are provided to the individual platforms. When a target passes near the platform, the time series data is process and matched against the reduced-dimensional vector database as described above and shown in the online SIF algorithm, Figure 8.8.

In practice, we extend the results of Bhatnagar with those of Wu *et al.* [10], which contains processing techniques designed to improve results for acoustic data. CPA event data are divided into training and test sets. The training data are used with the data-processing algorithm and the test data are used with the data-classification algorithm to evaluate the accuracy of the method. The training set is further divided into databases for each possible value of each target attribute being used in the

Let m = the number of target events, n = the number of channels per platform, p = the number of target attributes, and $\bar{T}^{ij}$ *= the time series for event i, channel j.*

For each target event *i*, and each sensor channel *j*

$\bar{s}^{ij} \leftarrow PowerSpectrum(\bar{T}^{ij})$; see [10, 11]

For each target event *i*

$\quad \bar{v}^i \leftarrow \bar{a}^i \,||\, \bar{s}^{i1} \,||\, \cdots \,||\, \bar{s}^{in}$

$M \leftarrow \bar{v}^1 \oplus \bar{v}^2 \oplus \cdots \oplus \bar{v}^m$

$(U, \Sigma, V^T) \leftarrow SVD(M)$; see [10, 12]

$k \leftarrow NumSignificantEignvalues(\Sigma)$

$\hat{U} \leftarrow TruncateColumns(\hat{U}, k)$

$\hat{V}^T \leftarrow TrunscateRows(V^T, k)$

$\hat{\Sigma} \leftarrow TruncateRows(TruncateColumns(\Sigma, k), k)$

Figure 8.7. SIF offline algorithm.

© 2005 by Chapman & Hall/CRC

Let m = the number of target events, n = the number of channels per platform, p = the number of target attributes, k = the reduced dimension size, and $\bar{T}^i$ = the time series from channel i.

For each sensor channel *i*

$\bar{s}^i \leftarrow PowerSpectrum(\bar{T}^i)$

$\bar{q} \leftarrow (0,\cdots,0) \parallel \bar{s}^1 \parallel \cdots \parallel \bar{s}^n$; the *m* components of the attribute vector ; are initially set to zero

$\hat{q} \leftarrow (\bar{q})^T \hat{U} \hat{\Sigma}^{-1}$

$r = index \max_i (p^i \cdot \hat{q}^i)$

$\hat{w} = MatrixColumn(V^T, r)$

$\bar{w} \leftarrow (\hat{w})^T \hat{\Sigma} \hat{U}$

$\bar{a}^q \leftarrow TrancateRows(\bar{w}, m)$; estimated attribute vector for target

Figure 8.8. Online SIF algorithm.

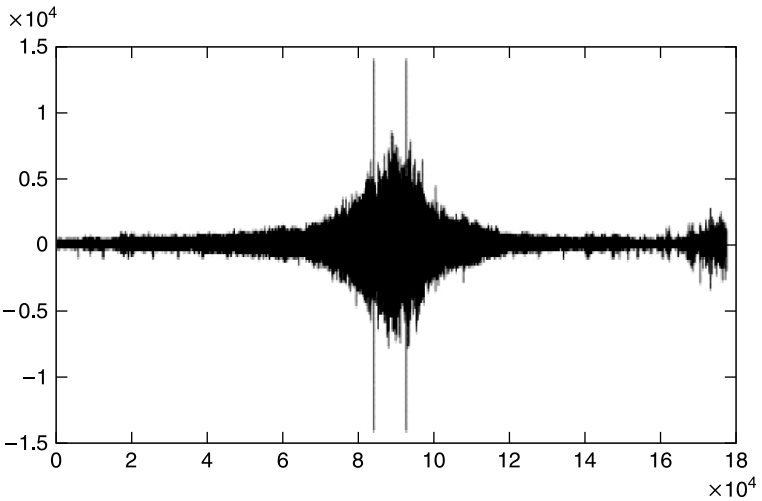


Figure 8.9. Time series window.

classification. Target attribute-values can be used to construct feature vectors for use in pattern classification. Alternatively, we can define "vehicle type" as a single attribute and identify the target directly.

A 4 to 5 s window is selected around the peak of each sample. All data outside of the window are discarded. This ensures that noise bias is reduced. The two long vertical lines in Figure 8.9 show what the boundaries of the window would be on a typical sample. The window corresponds to the period of time when a vehicle is closest to the platform. The data are divided into consecutive frames. A frame is 512 data points sampled at 5 KHz (0.5 s in length), and has a 12.5% (0.07 s) overlap with each of its neighbors. The power spectral density of each frame is found and stored as a column vector of 513 data points (grouped by originating sample), with data points corresponding to frequencies from 0 to 512 Hz.

Target identification combines techniques from Wu *et al.* [10] and makes use of an eigenvalue analysis to give an indication of the distance that an unknown sample vector is from the feature space of each database. This indication is called a residual. These residuals "can be interpreted as a measurement

© 2005 by Chapman & Hall/CRC

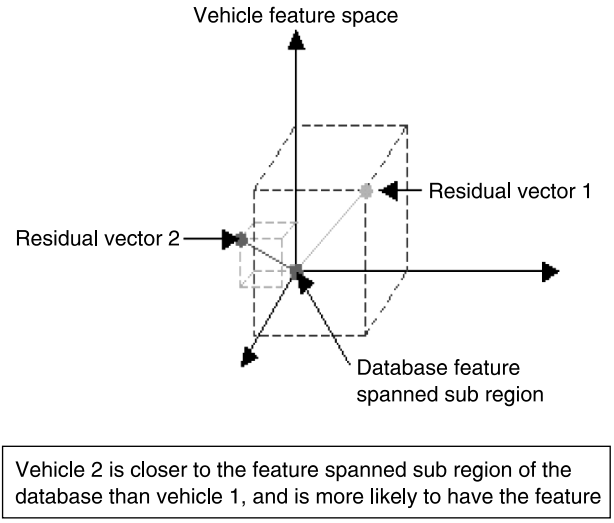


Figure 8.10. Isolating qualities in the feature space.

Table 8.2. Classification

Actual vehicle	Classified numbers			Correctly classified (%)
	AAV	DW	HV	
AAV	117	4	7	94
DW	0	106	2	98
HV	0	7	117	94

of the likelihood'' that the frame being tested belongs to the class of vehicles represented by the database [10]. The databases are grouped by attribute and the residuals of each frame within each group are compared. The attribute value corresponding to the smallest total of the residuals within each group is assigned to the frame. Figure 8.10 illustrates this process.

8.5.1 Experimental Results for SIF Classifier

Penn state applied research laboratory (ARL) evaluated its classification algorithms against the data collected during field tests. Data are shown for three types of military vehicle, labeled armored attack vehicle (AAV), Dragon wagon (DW), and hummvee (HV). The CPA peaks were selected by hand rather than automatically detected by the software, and there was only a single vehicle present in the network at a time. Environmental noise due to wind was significant. The data in Table 8.2 show that classification of military vehicles in the field can be accurate under noisy conditions.

8.6 Stationary Targets

8.6.1 Localization Using Signal Strengths

Another problem of interest for sensor networks is the counting and locating of stationary sources, such as vehicles with their engines running. Both theory and experiment suggest that acoustic energy for a single source is determined by $E = aJ/r^2$, where E is the energy measured by the sensor, r is the distance

© 2005 by Chapman & Hall/CRC

from the source to the sensor, J is the intensity of the source and a is approximately constant for a given set of synoptic measurements over the sensor network. Therefore $E_i(\bar{x}_s - \bar{x}_i)^2 = E_j(\bar{x}_s - \bar{x}_j)^2 \ \forall i,\ j \leq N$, where E_k is the energy measured by platform k, $\bar{x}_k$ is the location of platform k, N is the number of platforms in the network, and $\bar{x}_s$ is the location of the source. The location of the source is unknown, but it can be found iteratively by minimizing $\mathrm{Var}(E_i(\bar{x}_s - \bar{x}_i)^2)$ as a function of $\bar{x}_s$.

If node i is at (u, v) and the source is at (x, y), $r_i^2 = (x - u_i)^2 + (y - v_i)^2$, the equations for all the sensors can be represented in matrix form as

$$\frac{1}{aJ}\begin{bmatrix} E_1 & -2u_1 & -2v_1 & E(u_1^2 + v_1^2) \\ \vdots & \vdots & \vdots & \vdots \\ E_n & -2u_n & -2v_n & E(u_n^2 + v_n^2) \end{bmatrix}\begin{bmatrix} x^2 + y^2 \\ x \\ y \\ 1 \end{bmatrix} = I \tag{8.1}$$

where n is the number of sensors and I is the identity matrix. This over-determined set of equations can be solved for x, y, and J.

8.6.2 Localization Using Time Delays

We have also surveyed the literature to see whether signal time delays could be used in stationary-vehicle counting to enhance the results described above. A lot of work has been done in this area. We conclude that methods, as written, are not too promising when the vehicles are within the network, as opposed to the far field. The results and a suggestion for further research are summarized below.

The speed of sound in air is relatively constant. Therefore, given the delay between arrival times at multiple sensors of an acoustic signal from a single source, the distance from those sensors to the source can easily be calculated. Estimation of the location of that source then becomes a problem of triangulation, which, given enough microphones, can be considered an over-determined least-squares estimation problem. Thus, the problem of localization turns into one of time delay of arrival estimation.

When sources are far from the sensor network, the acoustic data can be thought of as arriving in an acoustic plane. The source is said to be in the far field. Finding the incidence of arrival of this plane and either enhancing or reducing its reception is the idea behind beam forming. When sources are close to the sensor array, the curvature of the surface of propagation through the array is pronounced; and now, more than an estimation of the direction of incidence of the wave front is required. The source is said to be in the near field, and is modeled as a single signal delayed by different amounts arriving at each sensor. Noise cancellation can be performed by removing any signals that arrive from the far field. Both of these topics are explored by Naidu [11].

Noise cancellation can be thought of as enhancing the signal of interest by finding its source in the near field. This is what we would like to use when trying to find the location of multiple vehicles in a sensor network. We want to find and count the various noise sources in the array. The task is difficult, and Emile *et al.* [12] claim that blind identification, in the presence of more than one source, when the signals are unknown, has only been solved well in the case of narrow-band spectra. There are many methods of performing time delay of arrival estimation, e.g. [13], and the main problem in the application of it to the problem of vehicle counting is selecting one that makes use only of the limited amount of information that we have of the array and environment. Even the size of the sample used to do the time delay of arrival may affect the performance significantly, as pointed out by Zou and Zhiping [14]. However, we may be able to make use of the idea of a local area of sensors, as we have in other algorithms such as the velocity estimator, to put bounds on the delay arrival times and the time the vehicle may be in the area. Therefore, we have information that may allow us to use one of the algorithms already in existence and remove enough of the errors associated with them to allow us to perform vehicle localization and/or counting in future work.

© 2005 by Chapman & Hall/CRC

8.6.3 Experimental Results for Localization Using Signal Strengths

Acoustic data were recorded from 20 sensors placed in a sensor mesh. Vehicles were driven into the mesh and kept stationary during the recording. Four tests were run, containing one, two, three, and three cars (in a different configuration than the third test). For example, test two had the layout shown in Figure 8.11. Figures 8.12 and 8.13 show how the data look for accurate and inaccurate estimates of $\bar{x}_s$. Figure 8.14 shows the dependence between estimates of $\bar{x}_s$ and $\mathrm{Var}(E_i(\bar{x}_s - \bar{x}_i)^2)$.

As shown in the Figure 8.14, we could resolve single vehicles to within the grid spacing of 20 ft. We could not, however resolve the multiple vehicle tests. For the two-vehicle test, the vehicles in Figure 8.11 were positioned at (50, 30) and (15, 15), approximately 38 ft apart. Figure 8.15 shows the resulting acoustic energy field. The peaks due to each vehicle cannot be resolved. Figure 8.16 shows the theoretical energy field derived from Equation (8.1) using the actual source locations and fitting the constant K to the experimental data. The two surfaces are similar, so we would not expect to resolve

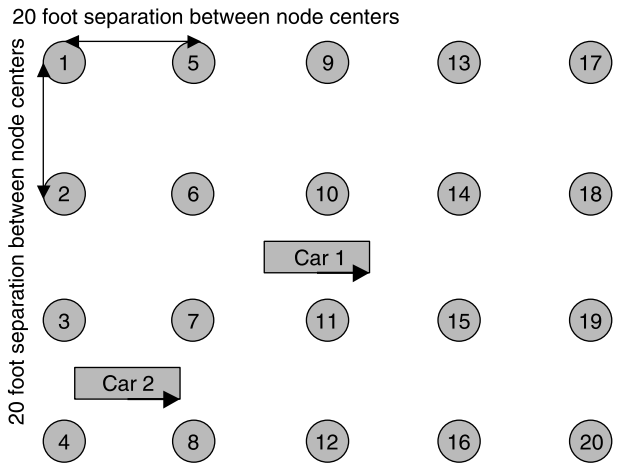


Figure 8.11. Experimental grid with two vehicles.

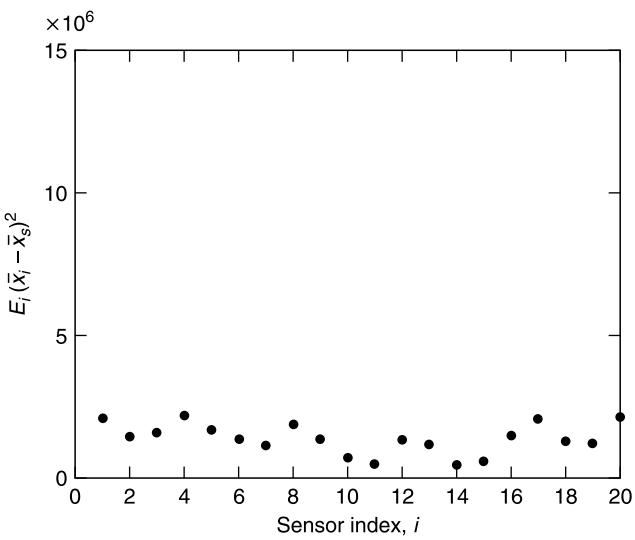


Figure 8.12. Data for an accurate location source estimate.

© 2005 by Chapman & Hall/CRC

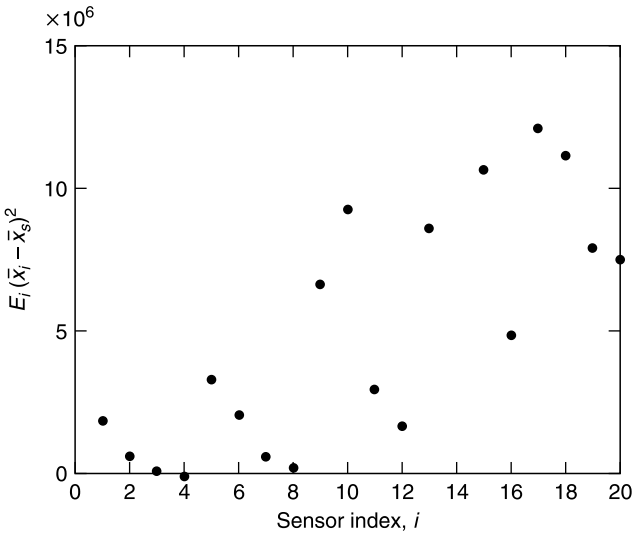


Figure 8.13. Data for an inaccurate source location estimate.

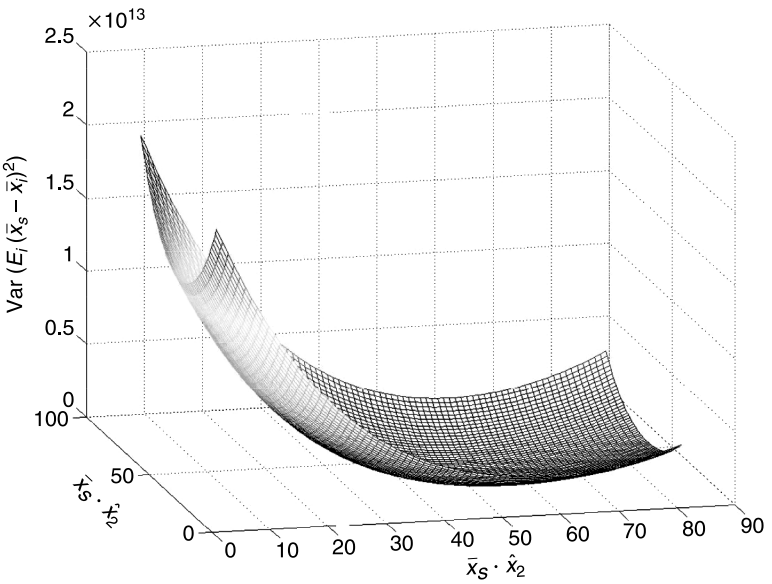


Figure 8.14. Minimization surface for a target located at (51, 31).

the two vehicles with a 20 ft grid. Figure 8.17 shows the theoretical energy field for a fine sensor grid. It suggests that a sensor grid of 10 to 15 ft would be needed to resolve the two vehicles, depending on where they were placed.

We conclude that single vehicles in a sensor grid can be detected and located to within the sensor separation distance and that multiple vehicles can be resolved to within three to four times the sensor separation distance.

In the first experiment, with a single car revving its engine, the location of the car can be found consistently in the correct location, (58, −30). It is marked in Figure 8.18, which also contains the acoustic sensor values when the engine is revving.

© 2005 by Chapman & Hall/CRC

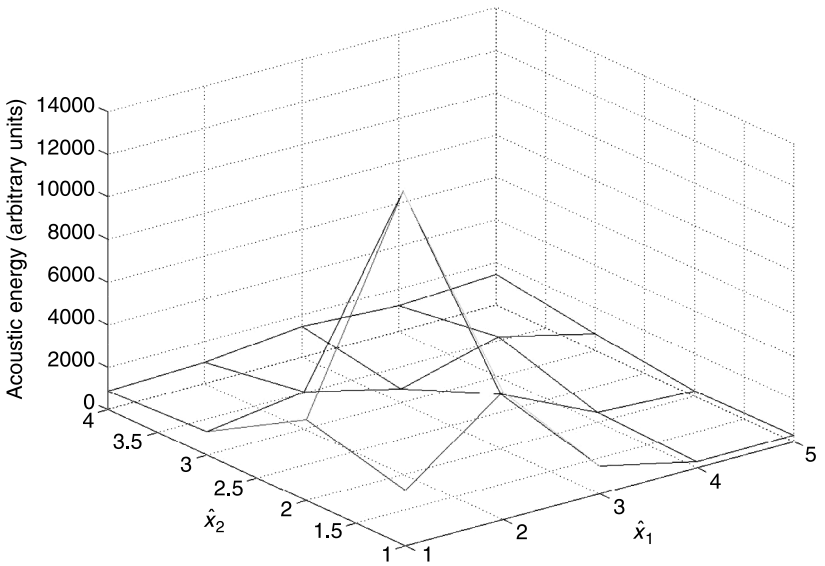


Figure 8.15. Acoustic energy field, two-vehicle test.

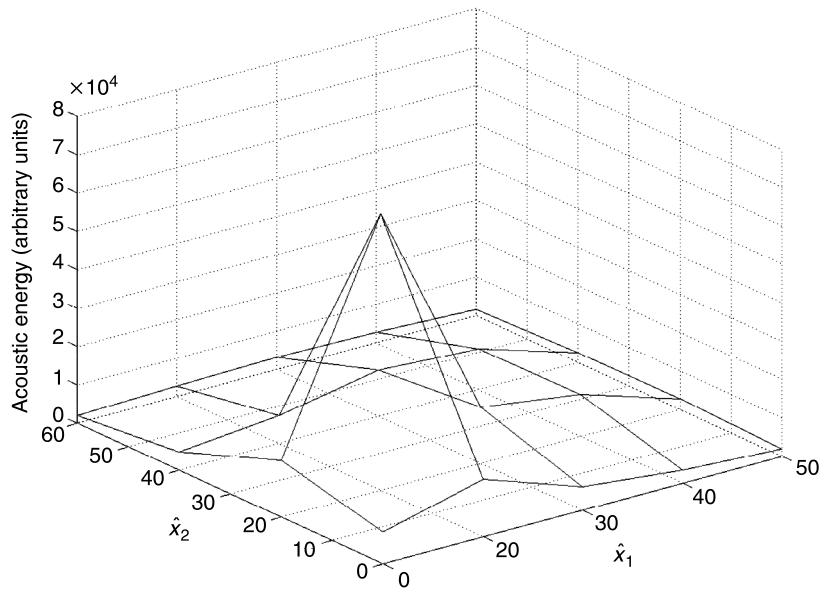


Figure 8.16. Theoretical energy field, 20 ft grid.

Figure 8.19 shows the acoustic data when the engine is not being revved. The location estimate is inaccurate. This follows from a geometric interpretation of the estimator. Effectively, a $1/r^2$ surface is fit to the data in the best way available in a least-squares error sense. Therefore, if noise or the propagation time of the acoustic energy warps the data far from the desired surface, then the estimate will be seemingly random.

When tracking vehicles we are only concerned when the CPA has been detected; therefore, the vehicle sound should have a large intensity, creating a large signal-to-noise ratio. Furthermore, only the sensors in the local area of the CPA need to be consulted, removing the noise of sensors far from the source. This work shows, however, that we may have trouble finding stationary idling vehicles.

© 2005 by Chapman & Hall/CRC

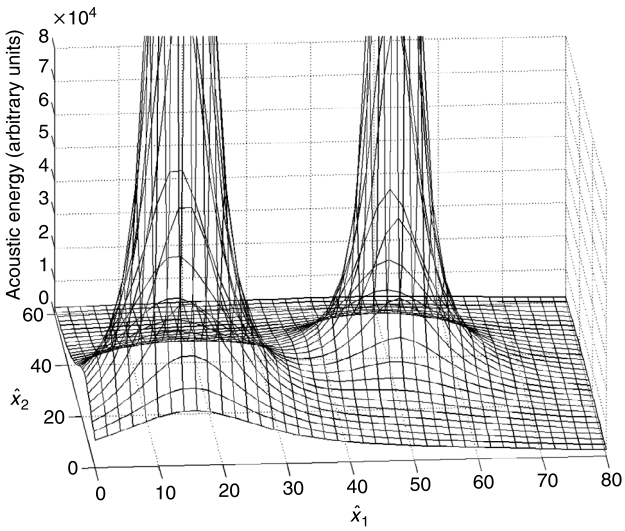


Figure 8.17. Theoretical energy field, 2 ft grid.

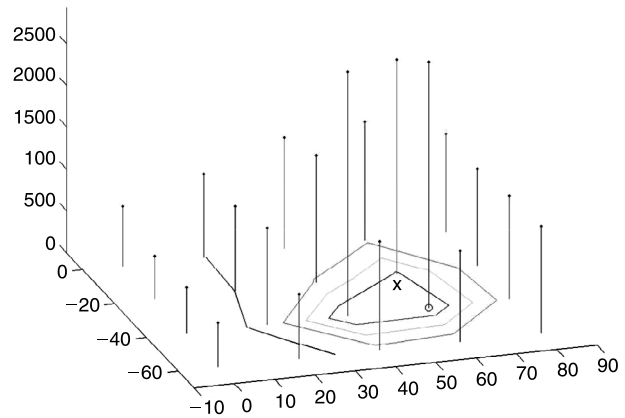


Figure 8.18. Acoustic sensor values and car location when engine is revving.

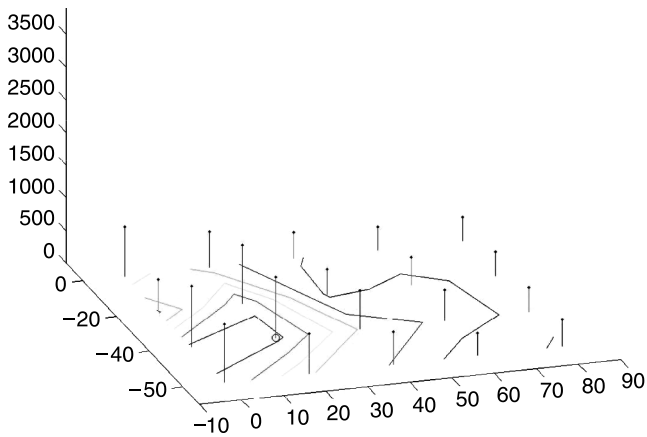


Figure 8.19. Acoustic data when the engine is not being revved.

© 2005 by Chapman & Hall/CRC

8.7 Peaks for Different Sensor Types

The quality of the local velocity determination algorithms is based in part on accurate determination of when a test vehicle is at its CPA to any given sensor node. We have begun analysis of how consistently this can be determined using three sensor modalities: acoustic, seismic and a two-pixel infrared camera. To facilitate this, we created a database of peak signal times for each node and sensor type. Three different military vehicles, labeled AAV, DW, and HV, are represented. There are database tables containing node information, global positioning system (GPS; ground truth), and CPA times. An overview of each table follows.

Node table. This table contains a listing of each node's UTM x and y locations (the unit is meters), indexed by node number. Figure 8.20 contains a plot of the node locations with the road overlaid.

GPS tables. These three tables contain the "ground truth" locations versus time. There is one table for each vehicle type. The GPS data were recorded every 2 s, and the second, minute, hour, day, month, and year recorded. The location of the vehicle at each time interval is recorded in two formats: UTM (the same as the node table) and latitude/longitude.

CPA table. We went through every acoustic, seismic, and passive infrared sensor file previously mentioned, and manually selected the peaks in each. Peak time is used to estimate the vehicle's CPA to the node that is recording the data. The CPA table contains records of the sensor, the vehicle causing the peak, the node associated with the peak, the peak time, and, for acoustic and seismic sensors, the maximum energy and amplitude of the signal.

Peaks for each sensor type were selected independently of one another. The CPA times for each sensor type tend to be similar; however, there are cases where one channel has a peak and the others do not. The data for node 4 are shown in Figure 8.21.

We have created a visualization of the combined CPA time and GPS data. The plot is three-dimensional, where the x and y axes are UTM coordinates and the z-axis is time in seconds since the first vehicle started. The path of the GPS device is the continuous line, and the dots are peak detections at the locations of the corresponding nodes. The "+" indicates a peak while the AAV was running, a "o" indicates a DW was running, and a "." indicates the HV runs. We have inclined the graph to get a good view of the situation, as shown in Figure 8.22. (The perspective results in a small

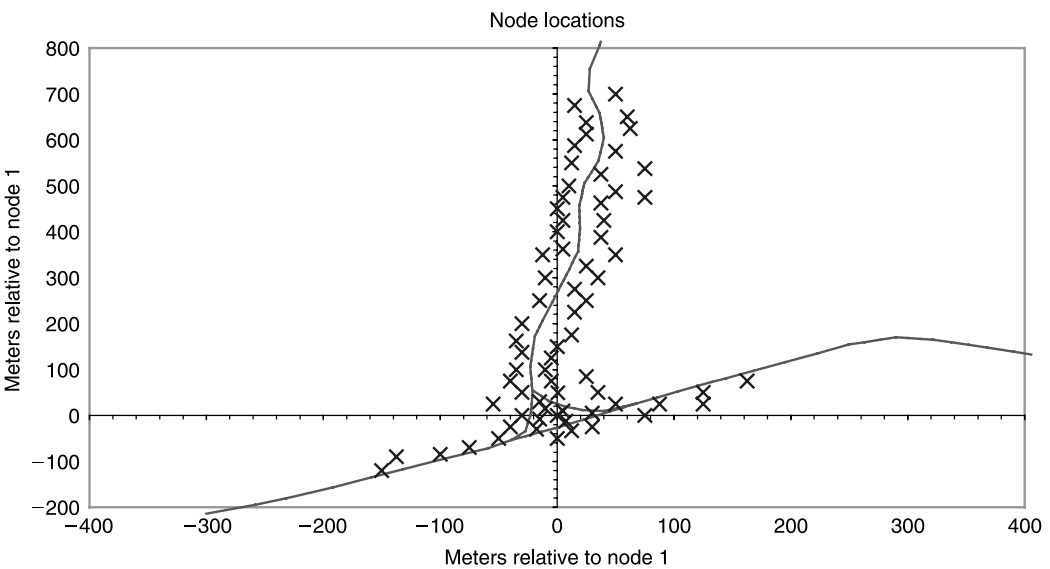


Figure 8.20. Node locations.

© 2005 by Chapman & Hall/CRC

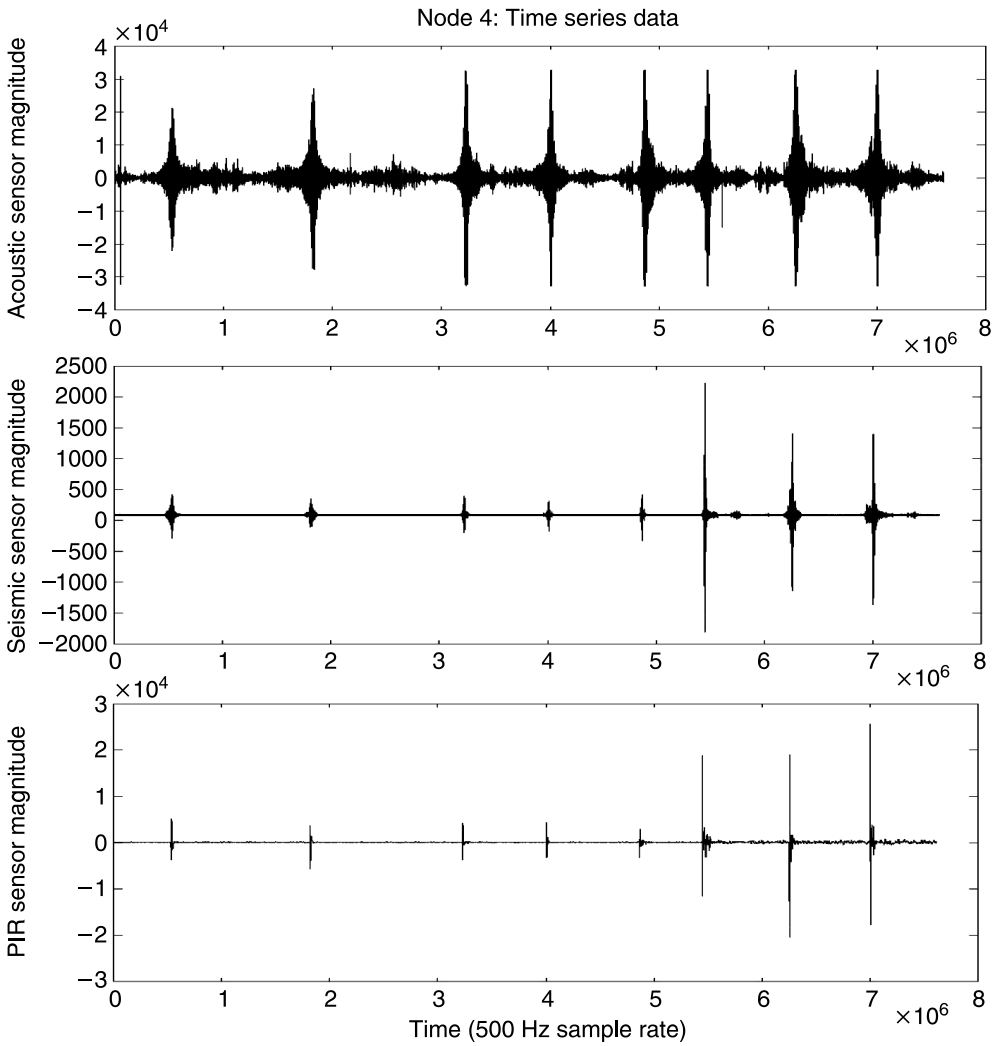


Figure 8.21. Infrared, seismic, and acoustic data.

display angle between the x and y axes.) A close-up of one of the vehicle traversals of the test region is shown in Figure 8.23.

All three sensors types do not always have a noticeable peak as a vehicle drives by. The peak database was used to examine the locality in time of the peaks from the different sensor types as a vehicle drives past a given node. We selected a time-window size (e.g. 10 s) and clustered all peaks at each specific node that occurred within the same window. This provided clusters of peaks for different sensor types that contained: just a single peak, two peaks, or three or more peaks. Figure 8.24 is a plot of the number of single, double and triple (or more) clusters versus the size of time window selected.

It usually takes 20–30 s for a vehicle to traverse the network area, and ideally there should only be one peak of each type at each node during each traversal, i.e. a single traversal results in one CPA event for each node. Thus, the statistics at this window size and larger should be relatively stable, because all peaks that are going to occur due to a given traversal should happen sometime during that 20–30 s period. The graph support this. However, a reduction of the window size down to 10–15 s yields little change. Hence, the data suggest that the peaks that are going to occur at a given node, from a single event, for the three sensor modalities usually occur within a 10–15 s window.

© 2005 by Chapman & Hall/CRC

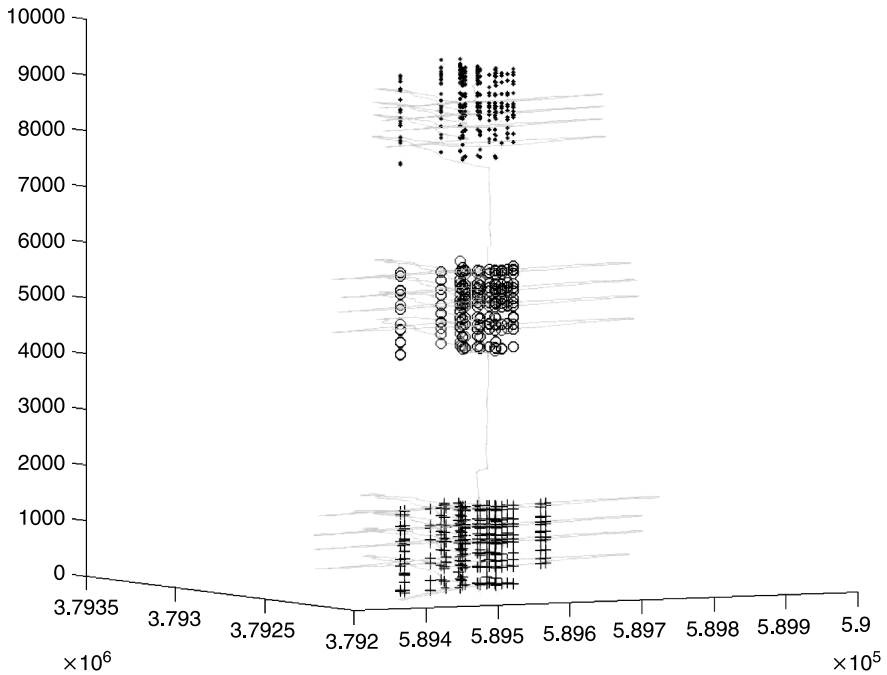


Figure 8.22. Three-dimentional plot of sensor peak and ground truth data.

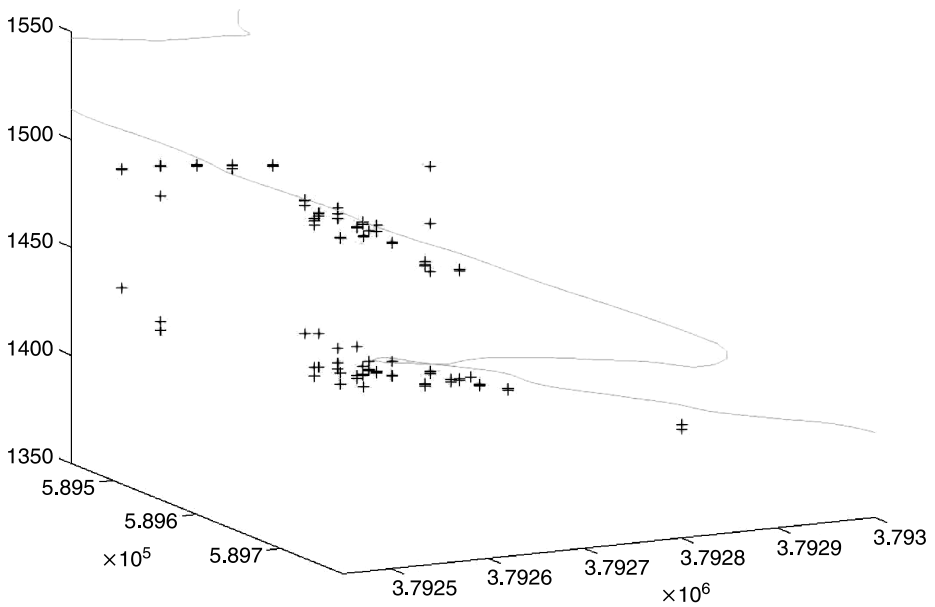


Figure 8.23. Close-up of AAV data.

© 2005 by Chapman & Hall/CRC

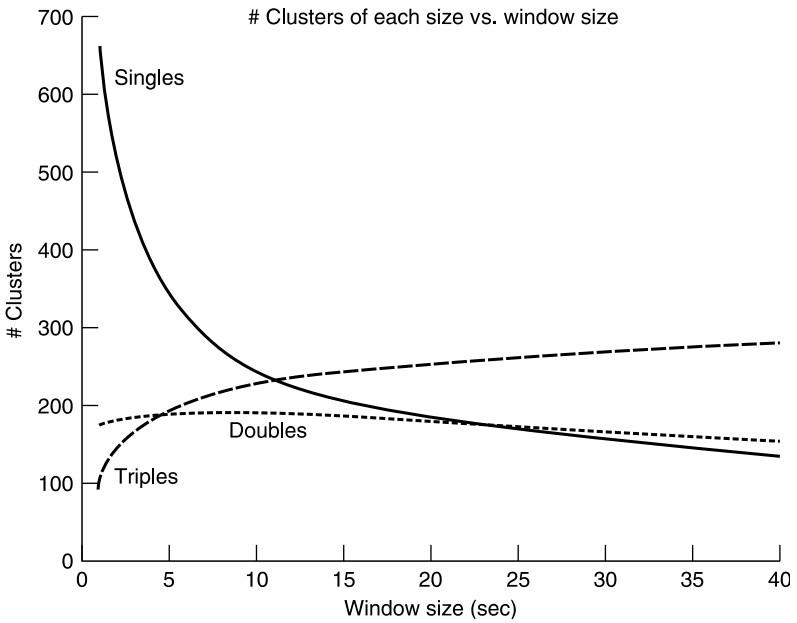


Figure 8.24. Sensor peak clusters vs. time window size.

Acknowledgments

This effort is sponsored by the Defense Advance Research Projects Agency (DARPA) and the Space and Naval Warfare Systems Center, San Diego (SSC-SD), under grant number N66001-00-C-8947 (Semantic Information Fusion in Scalable, Fixed and Mobile Node Wireless Networks), by the Defense Advance Research Projects Agency (DARPA) Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-99-2-0520 (Reactive Sensor Network), and by the US Army Robert Morris Acquisition under Award No. DAAD19-01-1- 0504.

The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the author's and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), the Space and Naval Warfare Systems Center, the Army Research Office, or the U.S. Government.

References

[1] Friedlander, D.S. and Phoha, S., Semantic information fusion of coordinated signal processing in mobile sensor networks, *Special Issue on Sensor Networks of the International Journal of High Performance Computing Applications*, 16(3), 235, 2002.

[2] Friedlander, D. *et al.*, Dynamic agent classification and tracking using an ad hoc mobile acoustic sensor network, *Eurasip Journal on Applied Signal Processing*, 2003(4), 371, 2003.

[3] Hellebrant, M. *et al.*, Estimating position and velocity of mobiles in a cellular radio network, *IEEE Transactions Vehicular Technology*, 46(1) 65, 1997.

[4] Phoha, S. *et al.*, Sensor network based localization and target tracking through hybridization in the operational domains of beamforming and dynamic space–time clustering, In *Proceedings of IEEE Global Communications Conference*, 1–5 December, 2003, San Francisco, CA, in press.

© 2005 by Chapman & Hall/CRC

[5] Brooks, R.R. *et al.*, Tracking targets with self-organizing distributed ground sensors, In *Proceedings to the IEEE Aerospace conference Invited Session "Recent Advances in Unattended Ground Sensors,"* March 10–15, 2003.
[6] Brooks, R. *et al.*, Distributed tracking and classification of land vehicles by acoustic sensor networks, *Journal of Underwater Acoustics*, in review, 2002.
[7] Brooks, R. *et al.*, Self-organized distributed sensor network entity tracking, *International Journal of High Performance Computer Applications*, 16(3), 207, 2002.
[8] Yao, K. *et al.*, Blind beamforming on a randomly distributed sensor array system, *IEEE Journal on Selected Areas in Communications*, 16, 1555, 1998.
[9] Jolliffe, I.T., *Principal Component Analysis*, Springer-Verlag, New York, 1986.
[10] Wu, H. *et al.*, Vehicle sound signature recognition by frequency vector principal component analysis, *IEEE Transactions on Instrumentation and Measurement*, 48(5), 1005, 1999.
[11] Naidu, P.S., *Sensor Array Signal Processing*, CRC Press LLC, New York, 2001.
[12] Emile, B. *et al.*, Estimation of time delays with fewer sensors than sources, *IEEE Transactions Signal Processing*, 46(7), 2012, 1998.
[13] Krolik, J. *et al.*, Time delay estimation of signals with uncertain spectra, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 36(12), 1801, 1988.
[14] Zou, Q. and Zhiping, L., Measurement time requirement for generalized cross-correlation based time-delay estimation, in *Proceedings of IEEE International Symposium on circuits and systems (ISCAS 2002)*, Vol. 3, Phoenix, USA, May 2002, 492.

© 2005 by Chapman & Hall/CRC

9

Target Tracking with Self-Organizing Distributed Sensors

R.R. Brooks, C. Griffin, David S. Friedlander, and J.D. Koch

9.1 Introduction

As computational devices have shrunk in size and cost, the Internet and wireless networking have become ubiquitous. Both trends are enabling technologies for the implementation of large-scale, distributed, embedded systems. Multiple applications exist for these systems in control and instrumentation. One area of particular importance is distributed sensing. Distributed sensing is necessary for applications of importance to government and industry, such as defense, transportation, border patrol, arms-control verification, contraband interdiction, and agriculture.

Sensors return information gained from physical interaction with their environment. To aid physical interaction, it is useful for sensors to be in close physical proximity with the objects observed. To observe actions occurring over a large region adequately, multiple sensors may be necessary. Battery-powered devices with wireless communications are advisable. It is also prudent to provide the devices with local intelligence.

It is rare for a sensor to measure information directly at the semantic level desired (e.g. how many cars have passed by this intersection in the last hour?). Sensors discern semantic information indirectly by interpreting entity interactions with the environment (e.g. cars are detected through acoustic vibrations emitted, or ground vibrations caused by wheels moving over terrain). Semantic information is inferred by interpreting one or more cues (often called features) detected by the sensor. Cues can be inexact and prone to misinterpretation. They contain noise and are sensitive to changes in the environment. Since the sensor has physical interaction with the environment, it is prone to failure, drift, and loss of calibration.

Creating a system from a large network of inexpensive intelligent sensors is an attractive means of overcoming these limitations. The use of multiple devices helps counter drift, component failure, and loss of calibration. It also allows for statistical analysis of data sources to filter noise better. Similarly, the use of multiple sensing modalities can make decisions more robust to environmental factors by increasing the number of cues available for interpretation.

© 2005 by Chapman & Hall/CRC

A central issue that needs to be overcome is the complexity inherent in creating a distributed system from multiple failure-prone components. Batteries run out. Components placed in the field are prone to destruction. Wireless communications are prone to disruption. Manual installation and configuration of nontrivial networks would be onerous. Similarly, programming networks and interpreting readings is time consuming and expensive.

In this chapter we present an example application that shows how to compensate for these issues. We discuss a flexible self-organizing approach to sensor network design, implementation, and tasking. The application described is entity tracking. Our approach decomposes the application into sub-tasks. Self-organizing implementations of each sub-task are derived. Example problems are discussed.

The rest of the chapter is organized as follows. Section 9.2 describes the computational environment used. Network interactions and self-organization use diffusion routing as implemented at USC/ISI [1] and a mobile code API implemented at the Penn State Applied Research Laboratory (ARL) [2]. The distributed multi-target tracking problem is decomposed into sub-problems in Section 9.3. In Section 9.4 we discuss the first sub-problem: how to use a cluster of nodes for collaborative parameter estimation. Once parameters have been estimated, they must be associated with tracks, as described in Section 9.5. Simulations comparing alternative approaches are described in Section 9.6.

Section 9.7 describes the use of cellular automata (CA) tools to evaluate and contrast network-embedded tracking approaches. Section 9.8 presents statistical and anecdotal results gained from using the CA models from Section 9.7 to study entity-tracking algorithms. Section 9.9 provides a brief description of the collaborative tracking network and some results of field tests. Sections 9.10 and 9.11 provide a comparative analysis of dependability and power requirements for our distributed tracking approach versus a more typical centralized approach. We describe the results of simulated multiple target tracking in Section 9.12. Section 9.13 concludes the chapter by presenting conclusions based on our work.

9.2 Computation Environment

This chapter describes three levels of work:

1. Theoretical derivations justify the approach taken.
2. Simulations provide proof-of-concept.
3. Prototype implementations give final verification.

We perform simulations at two levels. The first level uses in-house CA models for initial analysis. This analysis has been performed and is presented. Promising approaches will then be ported to the Virtual Internet Testbed (VINT) [3], which maintains data for statistical analysis. VINT also contains a network animation tool, *nam*, for visualization of network interactions. This supports replay and anecdotal study of pathologies when they occur. VINT has been modified to support sensor modeling.

In this chapter, we present mainly results based on simulations. Portions of the approach have been implemented and tested in the field. We differentiate clearly between the two. The hardware configuration of the prototype network nodes defines many factors that affect simulations and the final implementation. In this section we describe the prototype hardware and software environment, which strongly influences much of the rest of the chapter. For experimentation purposes, we use prototype sensor nodes that are battery-powered nodes with wireless communications. Each node has limited local storage and a CPU. For localization and clock synchronization, all nodes have global positioning system receivers. The sensor suite includes acoustic microphones, ground vibration detectors, and infrared motion detectors. All nodes have the same hardware configuration, with two exceptions: (i) the sensor suite can vary from node to node, and (ii) some nodes have more powerful radios. The nodes with more powerful radios work as gateways between the sensor network and the Internet. Development has been done using both Linux and Windows CE operating systems.

Wireless communications range is limited for several reasons. Short-range communications require less power. Since nodes will be deployed at or near ground level, multi-path fading significantly limits

© 2005 by Chapman & Hall/CRC

the effective range. The effective sensing range is significantly larger than the effective communications range of the standard radios.

These facts have distinct consequences for the resulting network topology. Short-range wireless communications make multi-hop information transmission necessary. Any two nodes with direct radio communications will have overlapping sensor ranges. The sensor field is dense. In most cases, more than one node will detect an event. Finite battery lifetimes translate into finite node lifetimes, so that static network configurations cannot be maintained.

Manual organization and configuration of a network of this type of any size would be a Sisyphean task. The system needs to be capable of self-configuration and automatic reconfiguration. In fact, the underlying structure of the network seems chaotic enough to require an *ad hoc* routing infrastructure. Static routing tables are eschewed in this approach. Routing decisions are made at run-time. It is also advisable to minimize the amount of housekeeping information transmitted between nodes, since this consumes power. Each bit transmitted by a node shortens its remaining useful lifetime.

To support this network infrastructure, a *publish–subscribe* paradigm has been used [1]. Nodes that are sources of information announce information availability to the network via a *publish* method provided by the networking substrate. When the information becomes available, a *send* method is used to transmit the information. Nodes that consume information inform the networking substrate of their needs by invoking a *subscribe* method. The *subscribe* method requires a parameter containing the address of a call-back routine that is invoked when data arrive.

A set of user-defined attributes are associated with each *publish* and *subscribe* call. The parameters determine matches between the two. For example, a *publish* call can have attributes whose values correspond to the UTM coordinates of its position. The corresponding *subscribe* would use the same attributes and define a range of values that include the values given in the *publish* call. Alternatively, it is possible to *publish* to a region and *subscribe* calls provide values corresponding to the UTM coordinates. It is the application programmer's responsibility to define the attributes in an appropriate manner. The *ad hoc* routing software establishes correspondences and routes data appropriately. Proper application of this *publish–subscribe* paradigm to the entity-tracking problem is an important aspect of this chapter. We use it to support network self-organization.

In addition to supporting *ad hoc* network routing, the system contains a mobile code infrastructure for flexible tasking. Currently, embedded systems have real constraints on memory and storage. This severely limits the volume of software that can be used by an embedded node, and directly limits the number of behaviors available. By allowing a node to download and execute code as required, the number of possible behaviors available can be virtually limitless. It also allows fielded nodes to be reprogrammed as required. Our approach manages code in a manner similar to the way a cache manages data. This encourages a coding style where mobile code is available in small packages.

In our approach, both code and data are mobile. They can be transferred as required. The only exceptions to this rule are sensors, which are data sources but tied to a physical location. We have implemented *exec* calls, which cause a mobile code package to execute on a remote node. Blocking and nonblocking versions of *exec* exist. Another important call is *pipe*. The semantics of this call are similar to a distributed form of pipes used by most Unix shell programs. The call associates a program on a node with a vector of input files and a vector of output files. When one of the input files changes, the program runs. After the program terminates, the output files are transmitted to other nodes as needed. This allows the network to be reprogrammed dynamically, using what is effectively an extensible distributed data-flow scripting language.

Introspective calls provide programs with information about mobile code modules resident on the network and on a specific node. Programs can pre-fetch modules and lock them onto a node. Locking a module makes it unavailable for garbage collection.

The rest of this chapter uses entity tracking as an example application for this sensor network computational environment. The environment differs from traditional approaches to embedded

© 2005 by Chapman & Hall/CRC

systems in many ways, specifically:

1. It is highly distributed.
2. It is assumed that individual components are prone to failure and have finite lifetimes.
3. Network routing is *ad hoc*.
4. The roles played by nodes change dynamically.
5. Sensing is done by collaboration between nodes.
6. A node's software configuration is dynamic.

These aspects of the system require a new programming approach. They also provide the system with the ability to adapt and modify itself when needed.

9.3 Inter-Cluster Tracking Framework

Sensor data interpretation takes place at multiple levels of abstraction. One example of this is the process from [4] shown in Figure 9.1. Sensor information enters the system. Objects are detected using signal-processing filters. Association algorithms determine which readings refer to the same object. Sequences of readings form tracks. The track information is used to estimate the future course of the entities and allocate sensors. The sensor allocation is done with a human in the loop guidance. We consider entity tracking as the following sequence of problems:

1. *Object detection*. Signal processing extracts features that indicate the presence of entities of interest.
2. *Object classification*. Once a detection event occurs, signal-processing algorithms assign the entity to one of a set of known classes. This includes estimating parameters regarding position, speed, heading, and entity attributes. Attributes can be combined into discrete disjoint sets that are often referred to as codebook values.
3. *Data association*. After classification, the entity is associated with a track. Tracks are initiated for newly detected entities. If a track already exists, then the new detection is associated with it. If it is determined that two separate tracks refer to the same entity, then they are merged.

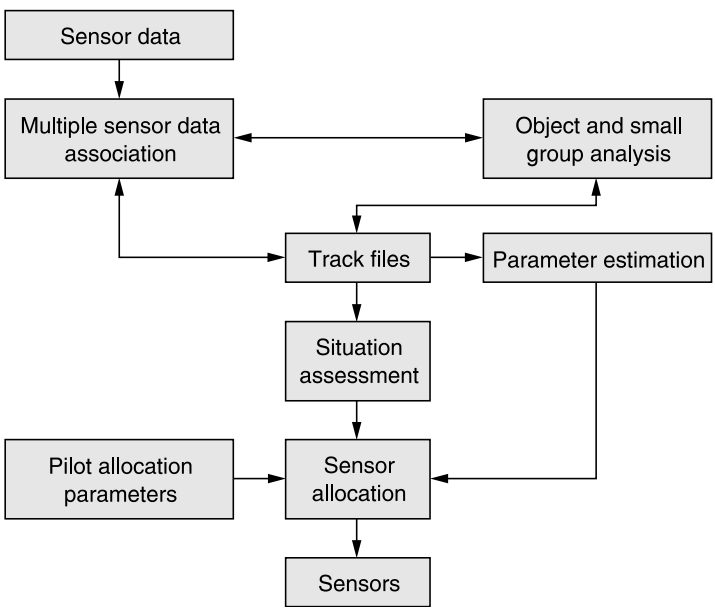


Figure 9.1. A concept for multiple sensor entity tracking from [4].

© 2005 by Chapman & Hall/CRC

4. *Entity identification*. Given track information, it may be possible to infer details about the identity and intent of an entity.
5. *Track prediction*. Based on current information, the system needs to predict likely future trajectories and cue sensor nodes to continue tracking the entity.

Object detection, classification, and parameter estimation will be discussed in Sections 9.3 and 9.4. Information is exchanged locally between clusters of nodes to perform this task. Data association, entity identification, and track prediction are all discussed in Section 9.5.

This chapter concentrates on embedding this sequence of problems defining entity tracking into the self-organizing network technologies described in Section 9.2. The approach given in this section supports multiple approaches to the individual problems. Figure 9.2 gives a flowchart of the logic performed at each node. The flow chart is not strictly correct, since multiple threads execute concurrently. It does, however, show the general flow of data through the system. In the current concept, each node is treated equally and all execute the same logic. This could be changed in the future.

The flowchart in Figure 9.2 starts with an initialization process. Initialization involves invoking appropriate *publish* and *subscribe* methods. *Subscribe* is invoked three times. One invocation has an associated parameter associating it with "near" tracks. One is associated with "mid" tracks. The third uses the parameter to associate it with "far" tracks. Figure 9.3 illustrates near, mid, and far regions. All three *subscribe* calls have two parameters that contain the node's x and y UTM coordinates. The *subscribe* invocations announce to the network routing substrate the node's intent to receive candidate tracks of entities that may pass through its sensing range. Near, mid, and far differ in the distance between the node receiving the candidate track and the node broadcasting candidate track information.

The flowchart in Figure 9.2 shows nodes receiving candidate tracks after initialization. For example, if the node in Figure 9.3 detects an entity passing by with a northeasterly heading, it will estimate the velocity and heading of the target, calculate and invoke publish to the relevant near, mid, and far regions, and invoke the network routing *send* primitive to transmit track information to nodes within those regions. Nodes in those regions can receive multiple candidate tracks from multiple nodes. The disambiguation process (see Figure 9.2) finds candidate tracks that are inconsistent and retains the tracks that are most likely. Section 9.5 describes example methods for performing this task. This step is important, since many parameter estimation methods do not provide unique answers. They provide a family of parallel solutions. It is also possible for more than one cluster to detect an entity. In the human retina and many neural network approaches, lateral inhibition is performed so that a strong response weakens other responses in its vicinity. We perform this function in the disambiguation task.

It is worth noting that the reason to *publish* (and *send*) to all three near, mid, and far regions shown in Figure 9.3 may be less than obvious. They increase system robustness. If no node is present in the near region or if nodes in the near region fail to detect an entity, the track is not necessarily lost.

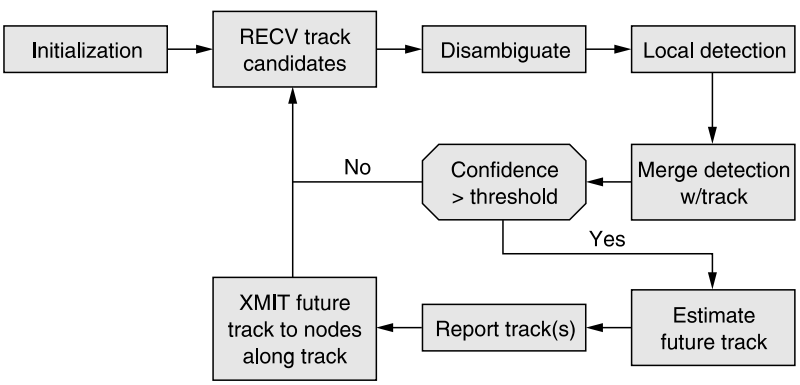


Figure 9.2. Flowchart of the processing performed at any given node to allow network-embedded entity tracking.

© 2005 by Chapman & Hall/CRC

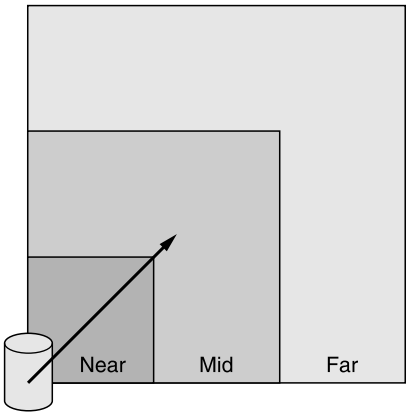


Figure 9.3. Example of dynamic regions used to *publish* candidate tracks. Solid arrow is the estimated target velocity and heading.

Nodes in the mid and far regions have candidate track information and may continue the track when an appropriate entity is detected. The existence of three levels (near, mid, and far) is somewhat arbitrary. Future research may indicate the need for more or fewer levels.

Track candidate reception and disambiguation run in one thread that produces an up-to-date list of tracks of entities that may be entering the local node's sensing range. Local detections refer to detection events within a geographic cluster of nodes. In Section 9.4 we explain how local information can be exchanged to estimate detection event parameters accurately, including position, heading, closest point of approach (CPA), and detection time.

When local detections occur, as detailed in Section 9.4, they are merged with candidates. Each track has an associated certainty factor. To reduce the number of tracks propagated by the system, a threshold is imposed. Only those tracks with a confidence above the threshold value are considered for further processing.

Fused tracks are processed to predict their future trajectory. This is similar to predicting the state and error covariance at the next time step given current information using the Kalman filter algorithm described by Brooks and Iyengar [5]. The predicted future track determines the regions likely to detect the entity in the future. The algorithm invokes the send method to propagate this information to nodes in those regions, thus completing the processing loop. Figure 9.4 shows how track information can be propagated through a distributed network of sensor nodes.

This approach integrates established entity tracking techniques with the self-organization abilities of the architecture described in Section 9.2. Association of entities with tracks is almost trivial in this approach, as long as sampling rates are high enough and entity distribution is low enough to avoid ambiguity. When that is not the case, local disambiguation is possible using established data association techniques [3]. A fully decentralized approach, like the one proposed here, should be more robust and efficient than current centralized methods. The main question is whether or not this approach consumes significantly more resources than a centralized entity tracking approach. This chapter is a first step in considering this problem.

9.4 Local Parameter Estimation

A number of methods exist for local parameter estimation. In August 2000 we tested a method of collaborative parameter estimation based on distributed computing fault tolerance algorithms given in Brooks and Iyengar [5]. Tests were run at the Marine Corps' Twenty Nine Palms test facility using the computational environment described in Section 9.2. Target detection was performed by signal-processing algorithms derived and implemented by BAE Systems Austin. Network routing used the ISI

© 2005 by Chapman & Hall/CRC

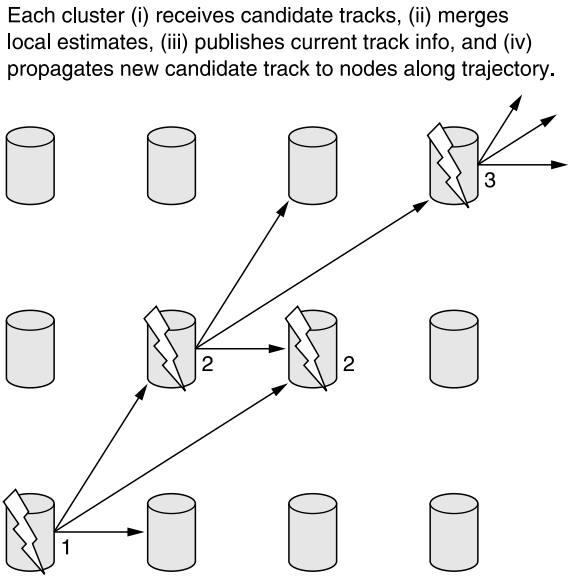


Figure 9.4. Example of how network-embedded entity tracking propagates track information in a network.

data diffusion methodology [1]. Penn State ARL implemented the collaboration and local parameter estimation approach described in this section. For this test, only acoustic sensor data was used.

Each node executed the *publish* method with three attributes: the node's UTM x coordinate, the node's UTM y coordinate, and a nominal value associating the *publish* with local collaboration. The same nodes execute the *subscribe* method with three attributes: a UTM range in the x dimension that approximates the coverage of the nodes acoustic sensor, a UTM range in the y dimension that approximates the coverage of the nodes acoustic sensor, and the nominal value used by the *publish* method.

When an entity is detected by a node, the *send* method associated with the *publish* transmits a data structure describing the detection event. The data structure contains the target class, node location, and detection time. All active nodes whose sensor readings overlap with the detection receive the data structure. The callback routine identified by their *subscribe* method is activated at this point. The callback routine invokes the local *send* method to transmit the current state of the local node. Temporal limits stop nodes from responding more than once.

In this way, all nodes exchange local information about the detection. One node is chosen arbitrarily to combine the individual detection events into a single collaborative detection. The test at Twenty Nine Palms combined readings at the node that registered the first detection. It has also been suggested that the node with the most certain detection would be appropriate. Since the same set of data would be merged using the same algorithm, the point is moot.

At Twenty Nine Palms the data were merged using the distributed agreement algorithm described Brooks and Iyenger [5]. This algorithm is based on a solution to the ''Byzantine Generals Problem.'' Arbitrary faults are tolerated as long as at least two-thirds of the participating nodes are correct and some connectivity restrictions are satisfied. The *ad hoc* network routing approach satisfies the connectivity requirements.

The algorithm uses computational geometry primitives to compute a region where enough sensors agree to guarantee the correctness of the reading in spite of a given number of possible false negatives or false positives. Figures 9.5(a)–(d) shows the results of this experiment. A sequence of four readings is shown. The entity was identified by its location during a given time window.

By modifying the number of sensors that had to agree, we were able to increase significantly the accuracy of our parameters. For each time slice, three decisions are shown proceeding clockwise from

© 2005 by Chapman & Hall/CRC

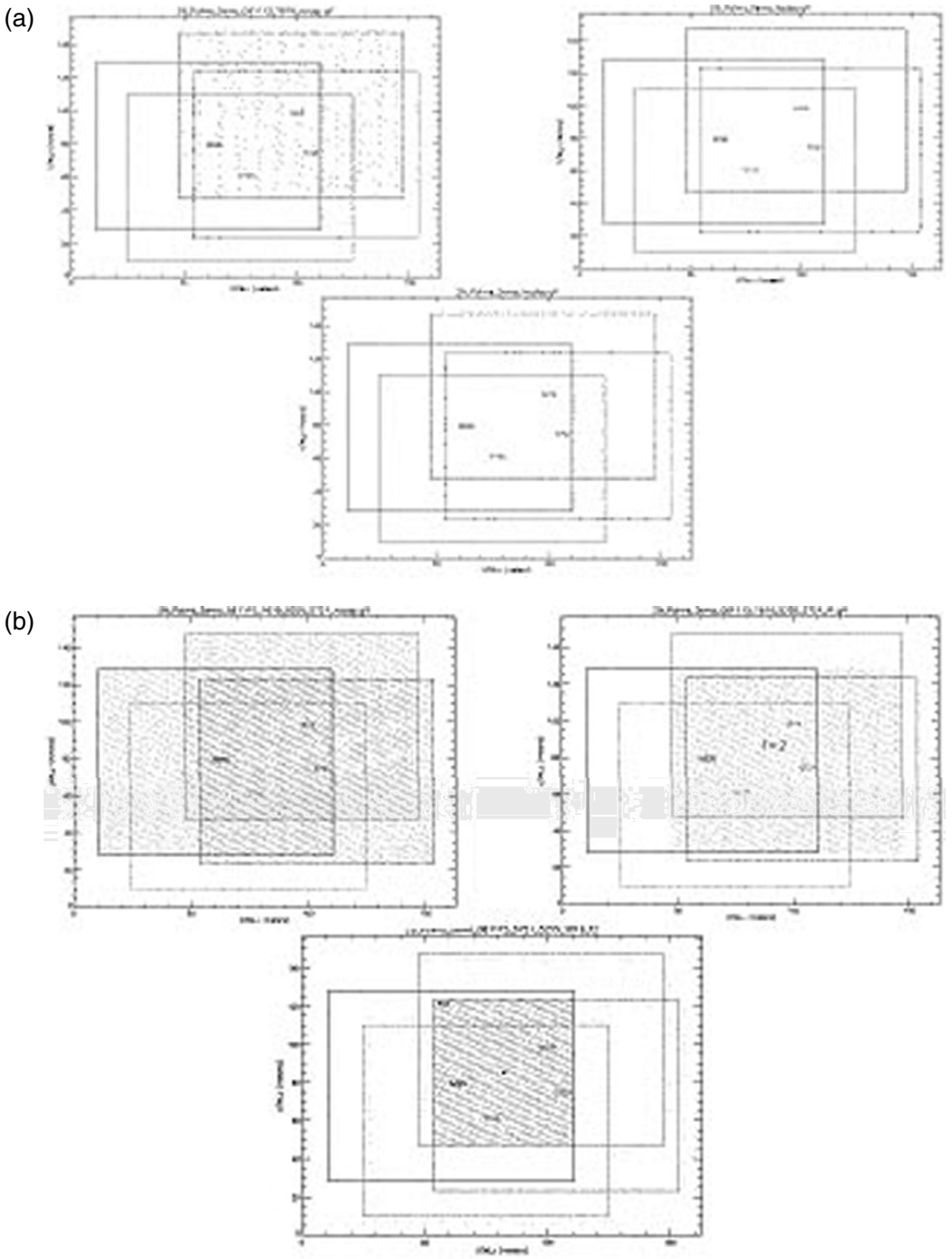


Figure 9.5. (a) The entity enters the sensor field from the northeast. It is within sensing range of only one sensor (1619). Since only one sensor covers the target, no faults can be tolerated. Clockwise from upper left, we show results when agreement is such that no fault, one fault, and two faults can be tolerated. (b) Same scenario as (a). The entity is within sensing range of three sensors (1619, 5255, 5721). Up to two faults can be tolerated. (c) Same scenario as (a). The entity is within sensing range of two sensors (1619, 5255). One fault can be tolerated. (d) Same as (a). The entity is within sensing range of two sensors (5255, 5721). One fault can be tolerated.

© 2005 by Chapman & Hall/CRC

(c)

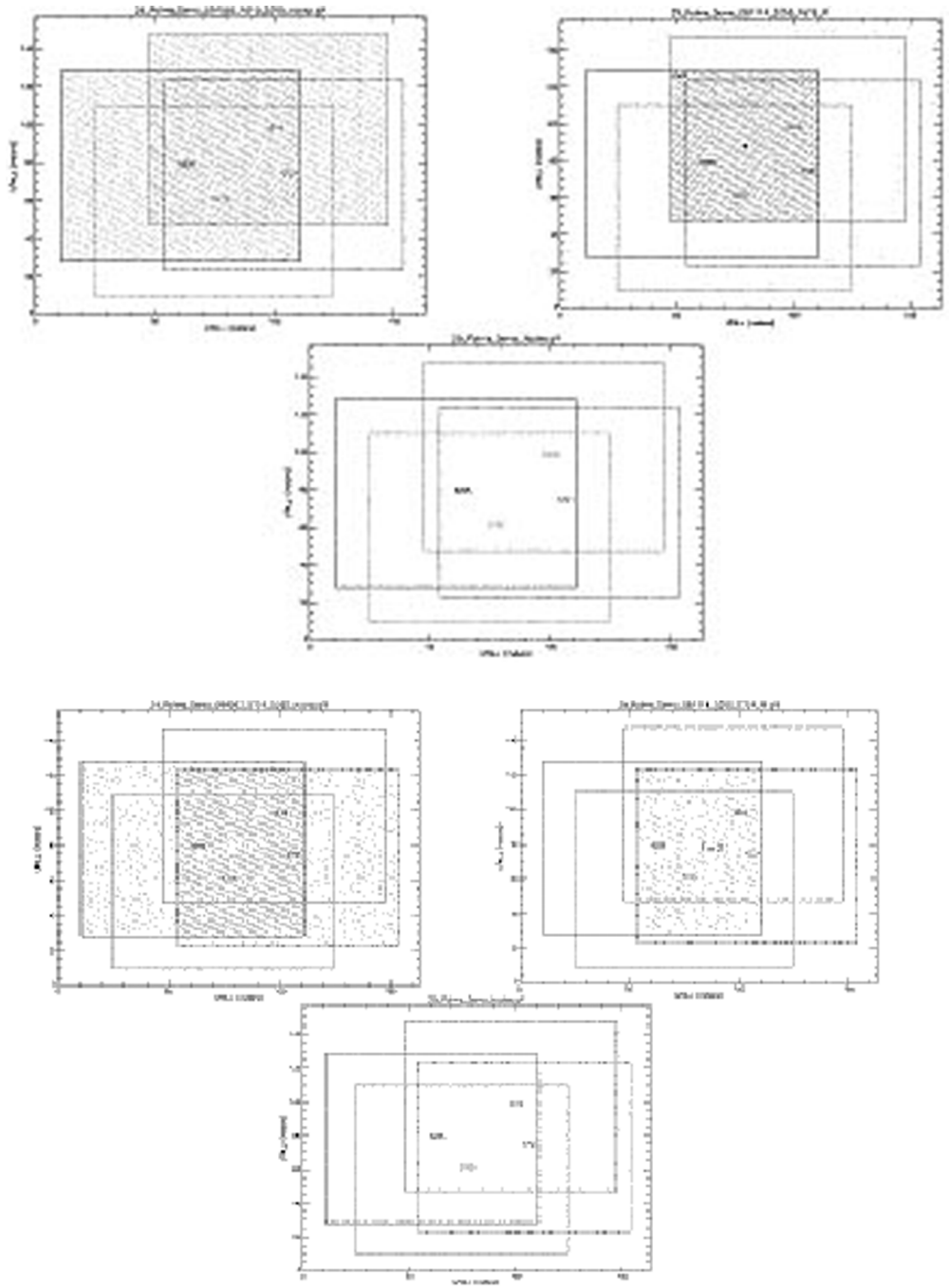


Figure 9.5. Continued.

the top left: (i) no faults tolerated, (ii) one fault tolerated, and (iii) two faults tolerated. Note that, in addition to increasing system dependability, the ability of the system to localize entities has improved greatly. It appears that localization improves as the number of faults tolerated increases. For a dense sensor network like the one used, this type of fault tolerance is useful and it is likely that, excluding boundary regions, entities will always be covered by a significant number of sensors.

© 2005 by Chapman & Hall/CRC

For many sensing modalities the sensor range can be very sensitive to environmental influences. For that reason it may be worthwhile using alternative statistics such as the CPA. The CPA can generally be detected as the point where the signal received from the entity is at its maximum. An alternative approach is being implemented, using the networking framework described in this section and CPA data. The CPA information is used to construct a trigonometric representation of the entity's trajectory. The solution of the problem returns information as to entity's heading and velocity. This approach is described in detail by Friedlander and Phoha [6]. Simulations indicate that this approach is promising. We tested it in the field in Fall 2001. The tracking methods given in Section 9.5 are designed using information from the CPA tracker, but they could also function using localization information from local collaboration.

9.5 Track Maintenance Alternatives

Given these methods for local collaboration to estimate entity-tracking parameters, we derive methods for propagating and maintaining tracks. Three separate methods will be derived: (i) pheromone tracking, (ii) extended Kalman filter (EKF), and (iii) Bayesian. All three methods are encapsulated in the "disambiguate," "merge detection with track," and "estimate future track" boxes in Figure 9.3. All three require three inputs: (i) current track information, (ii) current track confidence levels, and (iii) current parameter estimates. They produce three outputs (i) current best estimate, (ii) confidence of current best estimate, and (iii) estimated future trajectory. For each method in turn, we derive methods to:

1. Disambiguate candidate tracks.
2. Merge the local detection with the track information.
3. Initiate a new track.
4. Extrapolate the continuation of the track.

We now consider each method individually.

9.5.1 Pheromone Routing

For our initial approach, we will adapt pheromone routing to entity tracking. Pheromone routing is loosely based on the natural mechanisms used by insect colonies for distributed coordination [7].

When they forage, ants use two pheromones to collaborate in finding efficient routes between the nest and a food source. One pheromone is deposited by each ant, as it searches for food and moves away from the nest. This pheromone usually has its strongest concentration at the nest. Ants carrying food moving towards the nest deposit another pheromone. Its strongest concentration tends to be at the food source. Detailed explanations of exactly how these and similar pheromone mechanisms work are given by Brueckner [7].

Of interest to us is how pheromones can be an abstraction to aggregate information and allow information relevance to deteriorate over time, as shown in Figure 9.6. Pheromones are scent hormones that trigger specific behaviors. After they are deposited, they evaporate slowly and are dissipated by the wind. This means that they become less strong and more diffuse over time. This is useful for track formation. Entities move and their exact position becomes less definite over time. The relevance of sightings also tends to abate over time. The top portion of Figure 9.6 illustrates this. If two insects deposit the same pheromone in a region, the concentration of the pheromone increases. The sensory stimulation for other insects increases additively, as shown in the middle portion of Figure 9.6. Finally, multiple pheromones can exist. It is even possible for pheromones to trigger conflicting behaviors. In which case, as shown at the bottom of Figure 9.6, the stimuli can cancel each other providing less or no net effect.

These primitives provide a simple but robust method for distributed track formation. We consider each sensor node as a member of the insect society. When a node detects an entity of a specific type, it deposits pheromone for that entity type at its current location. We handle the pheromone as a random

© 2005 by Chapman & Hall/CRC

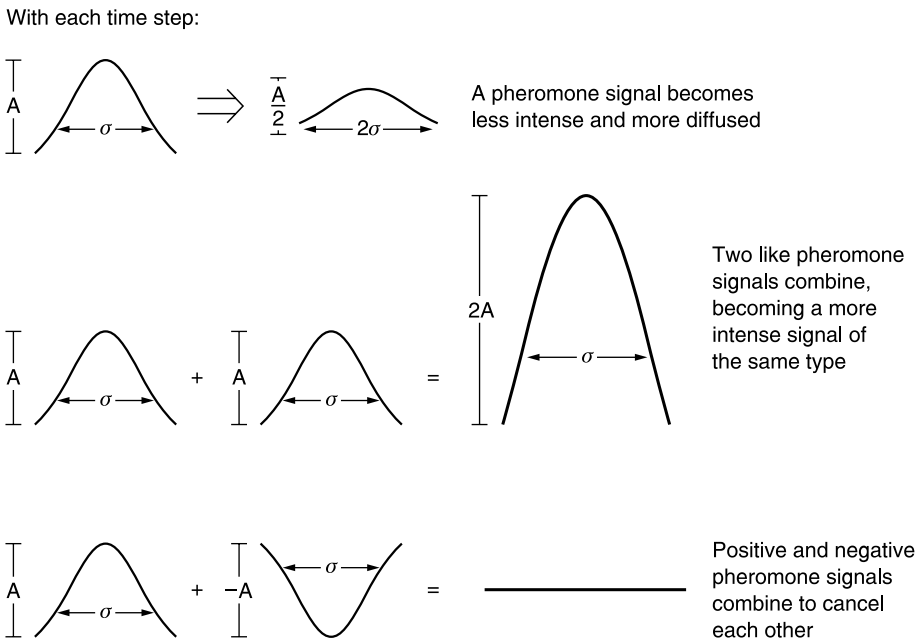


Figure 9.6. How pheromones can be used to diffuse (top), aggregate (middle), and cancel (bottom) information from multiple sources.

variable following a Gaussian distribution. The mean is at the current location. The height of the curve at the mean is determined by the certainty of the detection. The variance of the random variable increases as a function of time. Multiple detections are aggregated by summing individual detections. Note that the sum of normal distributions is a normal distribution.

Our method is only loosely based on nature. We try instead to adapt these principles to a different application domain. Track information is kept as a vector of random variables. Each random variable represents position information within a particular time window. Using this information we derive our entity tracking methods.

Track disambiguation performs the following steps for entity type and each time window:

1. Updates the random variables of pheromones attached to current track information by changing variances using the current time.
2. For each time window, sums the pheromones associated with that time window. Creates a new random variable representing the pheromone concentration. Since the weighted sum of a normal distribution is also normal, this maintains the same form.

This provides a temporal sequence of Gaussian distributions providing likely positions of each entity during increasing time windows.

Merging local detection with entity tracks is done by creating a probability density function for the current reading on the current node. The track is a list of probability density functions expressing the merged detections ordered by time windows.

Track initiation is done simply by creating a normal distribution that represents the current reading. This distribution is transmitted to nodes in the rectangular regions that are indicated by the heading parameter.

Track extrapolation is done by finding extreme points of the distribution error ellipses. For each node, four reference points are defined that define four possible lines. The region enclosed laterally by any two lines defines the side boundaries of where the entity is to be expected in the near future. The front and

© 2005 by Chapman & Hall/CRC

back boundaries are defined by the position of the local node (for far and near back), the time step used multiplied by the estimated velocity plus a safety factor (for near front), and twice the time step multiplied by the velocity estimate with safety (for far front). The pheromone information is transmitted to nodes in these two regions.

9.5.2 The EKF

This section uses the same approach as Brooks and Iyengar [5]. We will not derive the EKF algorithm here, as numerous control theory textbooks cover this subject in detail. We will derive the matrices for an EKF application that embeds entity tracking in the distributed sensor network. In deriving the equations we make some simple assumptions.

We derive a filter that uses the three most recent parameter estimates to produce a more reliable estimate. This filtering is required for several reasons. Differentiation amplifies noise. Filtering will smooth out the measurement noise. Local parameter estimates are derived from sensors whose proximity to the entity is unknown. Multiple answers are possible to the algorithms. Combining independently derived parameter estimates lowers the uncertainty caused by this.

When we collaboratively estimate parameters using local clusters of nodes, we assume that the path of the entity is roughly linear as it passes through the cluster. As described in Section 9.4, our velocity-estimation algorithm is based on the supposition that sensor fields will contain a large number of sensor nodes densely scattered over a relatively large area. Hence, we presume the existence of a sensor web, through which a vehicle is free to move. The algorithm utilizes a simple weighted least-squares regression approach, in which a parameterized velocity estimate is constructed.

As a vehicle moves through the grid, sensors are activated. Because the sensor grid is assumed to be densely positioned we may consider the position of an activated sensor node to be a good approximation of the position of the vehicle in question. Sensors are organized into families by spatial distance. That is, nodes within a certain radius form a family. The familial radii are generally small (6 m), so we may assume that the vehicle is moving in a relatively straight trajectory, free of sharp turns. If each sensor is able to estimate the certainty of its detection, then, as a vehicle moves through a clique of sensors, a set of four-tuples is collected:

$$D = \{(x_1, y_1, t_1, w_1), (x_2, y_2, t_2, w_2), \ldots, (x_n, y_n, t_n, w_n)\}$$

Each four-tuple consists of the UTM coordinates, (x_i, y_i) of the detecting sensor, the time of detection t_i, and the certainty of the detection $w_i \in [0, 1]$. Applying our assumption that the vehicle is traveling in linear fashion, we hope to fit the points to the equations

$$x(t) = v_x t + x_0$$

$$y(t) = v_y t + y_0$$

It may be assumed that nodes with a higher certainty were closer to the moving target than those with lower certainty. Therefore, we wish to filter out those nodes whose estimation of the target's position may be inaccurate. We do so by applying a weighted, linear regression [8] to the data above. The following equations show the result:

$$v_x = \frac{\sum_i w_i x_i \sum_i w_i t_i - \sum_i w_i \sum_i w_i x_i t_i}{\left(\sum_i w_i t_i\right)^2 - \sum_i w_i \sum_i w_i t_i^2}$$

$$v_y = \frac{\sum_i w_i y_i \sum_i w_i t_i - \sum_i w_i \sum_i w_i y_i t_i}{\left(\sum_i w_i t_i\right)^2 - \sum_i w_i \sum_i w_i t_i^2}$$

© 2005 by Chapman & Hall/CRC

The primary use of the resulting velocity information is position estimation and track propagation for use in the tracking system. Position estimation is accomplished using an EKF [5]. For the remainder of this paper we will use

$$\vec{x}_{k+1} = \Phi_k \vec{x}_k + \vec{w}_{k+1}$$

$$\vec{y}_k = \mathrm{M}_k \vec{x}_k + \vec{v}_k$$

as our filter equations. For our Kalman filter we have set

$$\vec{x}_k = \begin{pmatrix} x_k \\ y_k \\ v_k^x \\ v_k^y \end{pmatrix}$$

$$\Phi_k = \begin{pmatrix} 1 & 0 & \Delta t_k & 0 \\ 0 & 1 & 0 & \Delta t_k \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

and

$$\mathrm{M}_k = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & \Delta t_k & 0 \\ 0 & 1 & 0 & \Delta t_k \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & \Delta t_{k-1} & 0 \\ 0 & 1 & 0 & \Delta t_{k-1} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

where Δt_k is the time differential between the previous detection and the current detection. We are considering the last three CPA readings as the measurements. The covariance matrix of the error in the estimator is given by

$$P_{k+1} = \Phi_k P_k \Phi_k^{\mathrm{T}} + Q_{k+1}$$

where Q is the system noise covariance matrix. It is difficult to measure Q in a target-tracking application because there are no real control conditions. We have devised a method for estimating its actual value. The estimate, though certainly not perfect, has provided good experimental results in the laboratory. Acceleration bounding and breaking surfaces, for an arbitrary vehicle are shown in Figure 9.7.

We may construct an ellipse about these bounding surfaces, as shown in Figure 9.8. Depending upon the area of the ellipse and our confidence in our understanding of the vehicle's motion, we may vary the probability p that the target is within the ellipse, given that we have readings from within the bounding

© 2005 by Chapman & Hall/CRC

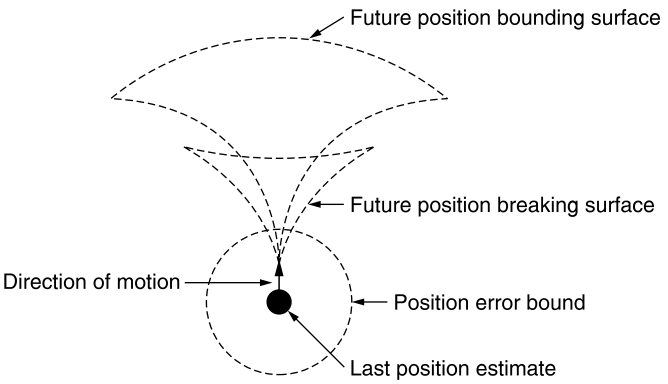


Figure 9.7. Target position uncertainty.

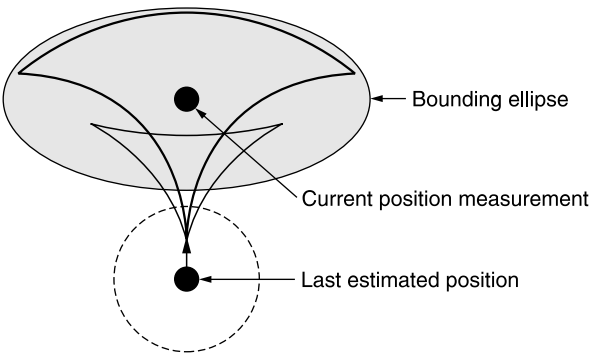


Figure 9.8. Target position bounding ellipse.

surfaces. We can use the radii of the ellipse and our confidence that our target is somewhere within this ellipse to construct an approximation for Q.

Assuming that the x and y values are independently and identically distributed, we have

$$p = \int_{-r_x}^{r_x} \exp\left[-\left(\frac{x}{\sigma_x\sqrt{2}}\right)^2\right]$$

Thus, we may conclude that

$$\sigma_x = \frac{r_x\sqrt{2}}{2_{p/2}}$$

and likewise that

$$\sigma_y = \frac{r_y\sqrt{2}}{2_{p/2}}$$

We can thus approximate a value for Q as

$$Q = \begin{pmatrix} \sigma_x & 0 & 0 & 0 \\ 0 & \sigma_y & 0 & 0 \\ 0 & 0 & \sigma_{v_x} & 0 \\ 0 & 0 & 0 & \sigma_{v_y} \end{pmatrix}$$

© 2005 by Chapman & Hall/CRC

In the matrix above, we do not mean to imply that these variances are actually independent, but this matrix provides us with a rough estimate for the noise covariance matrix, against which we can tune the Kalman filter. We have tested our velocity estimation and Kalman filter algorithms in the laboratory and at Twenty Nine Palms. The results of these tests are presented in the following sections.

9.5.3 Bayesian Entity Tracking

This section extends the Bayesian net concepts introduced by Pearl [9]. A belief network is constructed that connects beliefs that influence each other in the problem space. In this section we derive the portion of this belief network, which is embedded in single nodes in the sensor network. This portion receives inputs from other nodes in the sensor network. In this manner, the global structure of the extended Bayesian net extends across the sensor network and is not known in advance. The structure evolves in response to detections of objects in the environment. Although this evolution, and the computation of how beliefs are quantified, differs from the Bayesian net framework provided by Pearl [9], the basic precepts are the same.

Figure 9.9 shows the belief network that is local to a node in the sensor network. Candidate near and far tracks are received from other nodes in the network. The current net is set up to consider one near and one far candidate track at a time. This is described later in more detail. Detections refer to the information inferred by combining data from the local cluster of nodes. All three entities have a probabilistic certainty factor. They also have associated values for speed and heading. We use the same state vectors as in Section 9.5.1.

We will describe the functionality of the belief network from the bottom up:

- *No track* represents the probability that the current system has neither a new track nor a continuation of an existing one. It has the value $1 - (P_n + P_c - P_n P_c)$, where P_n is the probability that there is a new track and P_c is the probability that the detection is a continuation of an existing track. The sum of the probabilities assumes their independence.
- *New track* is the probability that a new track has been established. Its value is calculated by subtracting the likelihood that the current detection matches the near and far tracks under consideration from the certainty of the current detection.
- *Track continuation* expresses the probability that the current reading is a continuation of the near and far tracks under consideration. It is computed as $P_c = L_n + L_f - L_n L_f$, where P_c is the probability that the track is a continuation of the near track under consideration or the far track under consideration, and L_n (L_f) is the likelihood that the near (far) track matches the current detection.

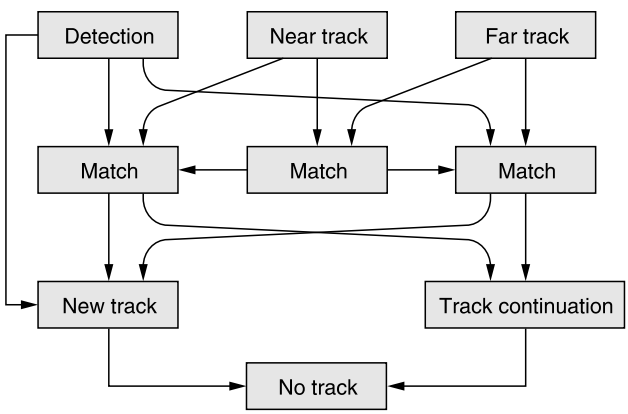


Figure 9.9. Belief network used in entity track evaluation.

© 2005 by Chapman & Hall/CRC

- *Matching* detections to tracks (or tracks to tracks) is done by comparing the value P_c with the variance of the track readings. This provides a probabilistic likelihood value for the current detection belonging to the current track. This value is weighted by the certainty values attached to the current detection L_d and track L_t. The weight we use is $1 - [(1 - L_t)(1 - L_d)]$, which is the likelihood that either the track or the detection are correct.
- *Matches* where a near and a far track match are favored by adding one-eighth of the matching value of the near and far tracks, as defined in the previous bullet point, to the value calculated above. The addition is done as adding two probabilities with the assumption of independence.

Given this belief network, we can now describe how the specific entity tracking methods.

Track disambiguation is performed by evaluating the belief network for every combination of near and far tracks. We retain the combination of tracks where the value of track continuation is a maximum. The decision is then made between continuing the tracks, starting a new track, or saying there is no current track by taking the decision with the highest probability.

Merging entity detection with local track is done by combining the current detection with the track(s) picked in the disambiguation step. If the match between the near and far tracks is significant, then the values of the near and far tracks are both merged with the current detection. If not, then the current detection is merged with the track it matches best. Merged parameters are their expected values. Parameter variance is calculated by assuming that all discrepancies follow a normal distribution.

Track initiation is performed when the decision is taken to start a new track during the disambiguation phase. Track parameters are the current estimate and no variance is available.

Track extrapolation is identical to the method given in Section 9.5.1.

9.6 Tracking Examples

In this section we present a simple example illustrating how each proposed method functions.

9.6.1 Pheromone Routing

Figure 9.10 represents a field of sensor nodes. An entity moves through the field and sensor clusters form spontaneously in response to detection events. Each cluster produces a pheromone

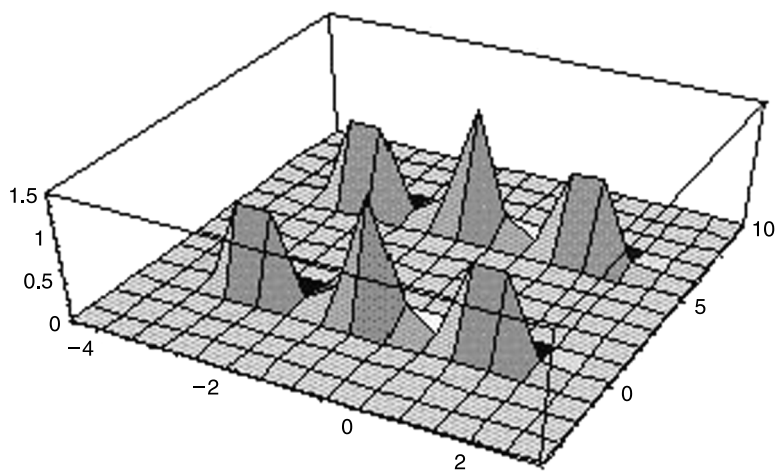


Figure 9.10. Entity detections from six sensor node clusters. The three detections at row 0 occur at time t_1. The three at row 5 occur at time t_2. Each detection results from a local collaboration. The pheromone is represented by a normal distribution.

© 2005 by Chapman & Hall/CRC

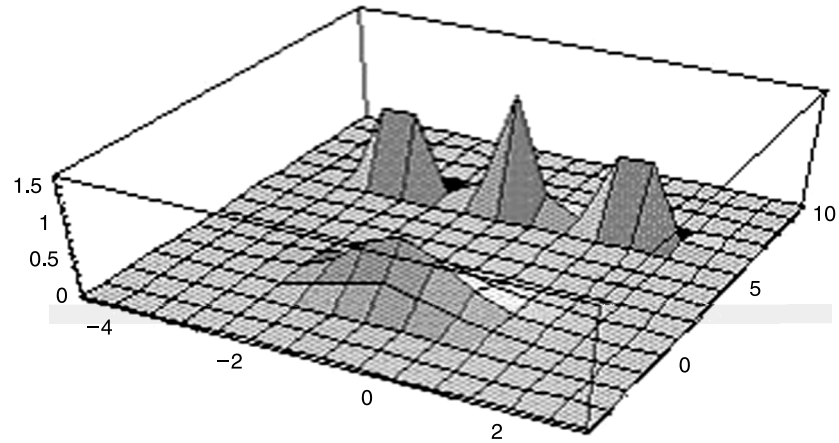


Figure 9.11. At time t_2 the pheromones exuded at time t_1 are combined into a single distribution.

abstraction signaling the presence of an entity. In this example, we consider a single class of entity and a single pheromone. Multiple pheromones could be used to reflect multiple entity classes, or entity orientation.

Figure 9.11 shows the situation at time t_2. Each cluster in row 5 exudes its own pheromone. They also receive pheromone information from the nodes in row 0. The three detections in row 0 were each represented as a random variable with a Gaussian distribution. The nodes performing the tracking locally independently combine the pheromone distributions from time t_1. This creates a single pheromone random variable from step one. This random variable encloses a volume equivalent to the sum of the volumes of the three individual pheromone variables at time t_1. The mean of the distribution is the sum of the means of the individual pheromone variables at time t_1. The new variance is formed by summing the individual variances from t_1 and increasing it by a constant factor to account for the diffusion of the pheromone.

The situation at time t_3 is shown in Figure 9.12. The pheromone cloud from time t_1 is more diffuse and smaller. This mimics the biological system, where pheromone chemicals evaporate and diffuse over time. Evaporation is represented by reducing the volume enclosed by the distribution. Diffusion is represented by increasing the variance.

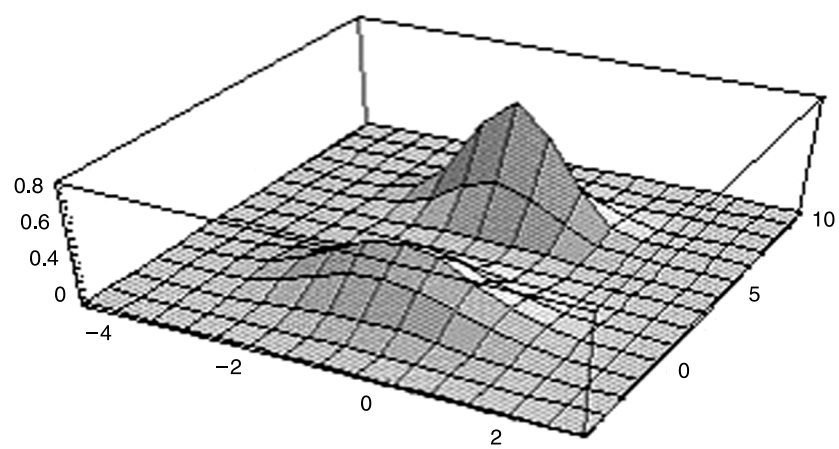


Figure 9.12. The situation at time t_3.

© 2005 by Chapman & Hall/CRC

9.6.2 The EKF

We present a numerical example of the EKF network track formation. Coordinates will be given in UTM instead of lat/long to simplify the example.

Without loss of generality, assume that our node is positioned at UTM coordinates(0, 0). We will not consider the trivial case when no detections have been made, but instead we assume that our node has been vested with some state estimate $\hat{\boldsymbol{x}}$ and some covariance matrix P from a neighboring node. Finally, assume that a target has been detected and has provided an observation y_0. Let

$$y = [\langle 0.2,\ 3.1,\ 0.1,\ 3.3\rangle, \langle -0.5,\ 2.8,\ -0.6,\ 2.7\rangle, \langle -0.8,\ 3,\ 0,\ 2.5\rangle], \quad \hat{\boldsymbol{x}}(k/k) = [-0.225,\ 3.0,\ 0.17,\ 2.83]$$

and assume $y_0 = [0, 3.1, 0, 2.9]$. Finally, let

$$P(k|k) = \begin{bmatrix} 0.4 & 0 & 0 & 0 \\ 0 & 1.1 & 0 & 0 \\ 0 & 0 & 0.3 & 0 \\ 0 & 0 & 0 & 1.2 \end{bmatrix}$$

be the covariance matrix last computed. Since there is only one track it is clear that the minimum of $\sqrt{(y_0 - \hat{\boldsymbol{x}})^2}$ will match the the state estimate given above. As soon as detection occurs, y becomes

$$y = [\langle 0,\ 3.1,\ 0,\ 2.9\rangle, \langle 0.2,\ 3.1,\ 0.1,\ 3.3\rangle, \langle -0.5,\ 2.8,\ -0.6,\ 2.7\rangle]$$

We can now compute $P(k+1|k+1)$ and $\hat{\boldsymbol{x}}(k+1|k+1)$; from Section 9.5.2 we have

$$\hat{\boldsymbol{x}}(k+1|k+1) = [-0.2869,\ 3.1553,\ -0.293,\ 2.8789]$$

and

$$P(k+1|k+1) = \begin{bmatrix} 0.15 & 0.21 & 0.05 & 0.02 \\ 0.16 & -0.07 & 0.2 & -0.07 \\ -0.08 & 0.11 & 0.17 & 0.18 \\ 0.21 & 0.05 & 0.05 & 0.22 \end{bmatrix}$$

This information, along with the last observation can now be sent to nodes in the direction of the vehicle's motion, namely in a northeastwardly direction heading away from (0, 0) towards (1, 1) in UTM.

9.6.3 Bayesian Belief Net

We will construct a numerical example of Bayesian network track formation from the node point of view. The values given will be used in the Section 9.7 for comparing the tracking approaches derived here. Further research is needed to determine appropriate likelihood functions empirically. These values are used for initial testing. Without loss of generality, assume that

$$L_{\mathrm{n}} = L_{\mathrm{f}} = \begin{cases} 0.5 & \text{if a detection exists} \\ 0 & \text{otherwise} \end{cases}$$

© 2005 by Chapman & Hall/CRC

where L_n (L_f) is the likelihood that a near (far) track matches a current detection. Also, let

$$L_t = 1 - L_d = \begin{cases} 0.5 & \text{if a detection exists} \\ 0 & \text{otherwise} \end{cases}$$

Assume that detection has occurred in a node with no near or far track information; then, the following conclusions may be made:

$$P_c = L_n + L_f - L_n L_f = 0$$
$$P_{No} = 1 - P_n$$

and P_{No} is precisely the probability of a false positive. Assuming perfect sensors, then $P_n = 1$ and the node begins a new track.

Assume that a node detects a target and has near and far track information; then, the following conclusions can be drawn:

$$P_c = L_n + L_f - L_n L_f = 0.75$$

$$P_m = 1 - (1 - L_t)(1 - L_d) = 0.75$$

Therefore, there is a 75% chance that this is a track continuation with a confidence of 75% that the match is correct. Assuming a perfect sensor, $P_n = 0.25$, since no false detections can be made.

9.7 The CA Model

Evolving distributed systems have been modeled using CA [12]. CA are a synchronously interacting sets of abstract machines (network nodes). CA are defined by:

- d the dimension of the automata
- r the radius of an element of the automata
- δ the transition rule of the automata
- s the set of states of an element of the automata

An element's (node's) behavior is a function of its internal state and those of neighboring nodes as defined by δ. The simplest instances of CA have a dimension of 1, a radius of 1, a binary set of states, and all elements are uniform. In this case, for each individual cell there are a total of 2^3 possible configurations of a node's neighborhood at any time step — if the cell itself is considered part of its own neighborhood. Each configuration is expressed as an integer v:

$$v = \sum_{i=-1}^{1} 2^{j_i + 1} \tag{9.1}$$

where i is the relative position of the cell in the neighborhood (left: -1; current position: 0; right: 1), and j_i is the binary value of the state of cell i. Each transition rule can, therefore, be expressed as a single integer r known as its Wolfram number [11]:

$$r = \sum_{v=1}^{8} j_v \times 2^v \tag{9.2}$$

© 2005 by Chapman & Hall/CRC

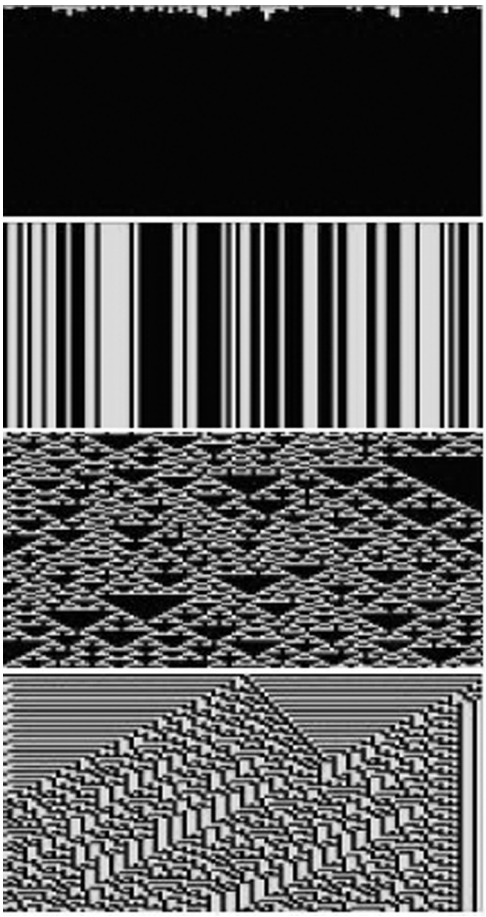


Figure 9.13. Examples of the four complexity classes of CA. From the top: (i) uniform, (ii) periodic, (iii) chaotic, and (iv) interesting.

where j_v is the binary state value for the cell at the next time step if the current configuration is v. This is the most widely studied type of CA. It is a very simple many-to-one mapping for each individual cell.

The four complexity classes shown in Figure 9.13 have been defined for these models. In the uniform class, all cells eventually evolve to the same state. In the periodic class, cells evolve to a periodic fixed structure. The chaotic class evolves to a fractal-like structure. The final class shows an interesting ability to self-organize into regions of local stability. This ability of CA models to capture emergent self-organization in distributed systems is crucial to our study.

We use more complex models than those given by Equations (9.1) and (9.2). CA models have been used successfully to study traffic systems and mimic qualitative aspects of many problems found in vehicular traffic flow [12]. For example, they can illustrate how traffic jams propagate through road systems. By modifying system constraints, it is possible to create systems where traffic jams propagate either opposed to or along the direction of traffic flow. This has allowed physicists to study empirically how highway system designs influence the flow of traffic.

Many of the CA models are called ''particle-hopping'' models. The most widespread particle-hopping CA model is the Nagel–Schreckenberg model [12]. This is a variation of the one-dimensional CA model [10] expressed by Equations (9.1) and (9.2). This approach mainly considers stretches of highway as one-dimensional CA. It typically models one lane of a highway. The highway is divided into sections, which are typically uniform. Each section of the highway is a cell. The sizes of the cells are such that the

© 2005 by Chapman & Hall/CRC

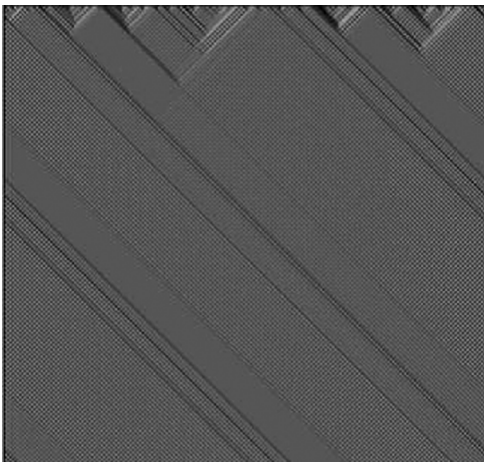


Figure 9.14. Example of output from particle-hopping CA. Lighter shades of gray signal higher packet density. This is a one-dimensional example. The x dimension is space. Each row is a time step. Time evolves from top to bottom. Black diagonal stripes from top left to bottom right show caravan formation. Light stripes from right to left at the top of the image show traffic jam propagation.

state of a cell is defined by the presence or lack of an automobile in the cell. All automobiles move in the same direction.

With each time step, every cell's state is probabilistically defined based on the states of its neighbors. Nagel–Schreckenberg's CA is based on mimicking the motion of an automobile. Only one automobile can be in a cell at a time, since two automobiles simultaneously occupying the same space causes a collision. If an automobile occupies a cell, then the probability of the automobile moving to the next cell in the direction of travel is determined by the speed of the automobile. The speed of the automobile depends on the amount of free space in front of the automobile, which is defined by the number of vacant cells in front of the automobile. In the absence of other automobiles (particles), an automobile moves at maximum speed along the highway by hopping from cell to cell. As more automobiles enter the highway, congestion occurs. The distance between particles decreases, and consequently the speed decreases. Figure 9.14 shows the evolution of a particle-hopping CA over time.

We adapt this approach to modeling sensor networks. Instead of particles representing automobiles moving along a highway, they represent packets in a multi-hop network moving from node to node. Each cell represents a network node rather than a segment of a highway lane. Since we are considering a two-dimensional surface covered with sensor nodes, we need two-dimensional CA. The cells are laid out in a regular matrix. A node's neighborhood consists of the eight nodes adjoining it to the north, south, east, west, northwest, northeast, southwest and southeast. For this paper we assume that nodes are fixed geographically, i.e. non-mobile. A packet can move from a node to any of its neighbors. The number of packets in the cell's node defines the cell's state. Each node has a finite queue length. A packet's speed does not depend on empty cells in its vicinity. It depends on the node's queue length. Cell state is no longer a binary variable; it is an integer value between 0 and 10 (chosen arbitrarily as the maximum value).

As with Nagel–Schreckenberg, particle (packet) movement from one cell to another is probabilistic. This mirrors the reality that wireless data transmission is not 100% reliable. Atmospheric and environmental affects, such as sunspots, weather, and jamming can cause packets to be garbled during transmission. For our initial tests, we have chosen the information sink to be at the center of the bottom edge of the sensor field. Routing is done by sending packets along the shortest viable path from the sensor source to the information sink, which can be determined using local information. Paths are not viable when nodes in the path can no longer receive packets. This may happen when a node's battery is exhausted or its queue is full.

© 2005 by Chapman & Hall/CRC

This adaptation of particle-hopping models is suitable for modeling the information flow in the network; however, it does not adequately express sensing scenarios where a target traverses the sensor field. To express scenarios we have included "free agents in a cellular space" (FACS) concepts [11].

Portugali [11] used ideas from synergetics and CA including agents to study the evolution of ethnic distributions in Israeli urban neighborhoods. In the FACS model, agents are free to move from cell to cell in the CA. The presence of an agent modifies the behavior of the cell, and the state of a cell affects the behavior of an agent.

In our experiments, entities traversing the sensor field are free agents. They are free to follow their own trajectory through the field. Detection of an entity by a sensor node (cell) triggers one of the entity-tracking algorithms. This causes track information to be transmitted to other nodes and to the information sink. Figure 9.15 describes the scenarios we use in this paper to compare the three tracking approaches proposed.

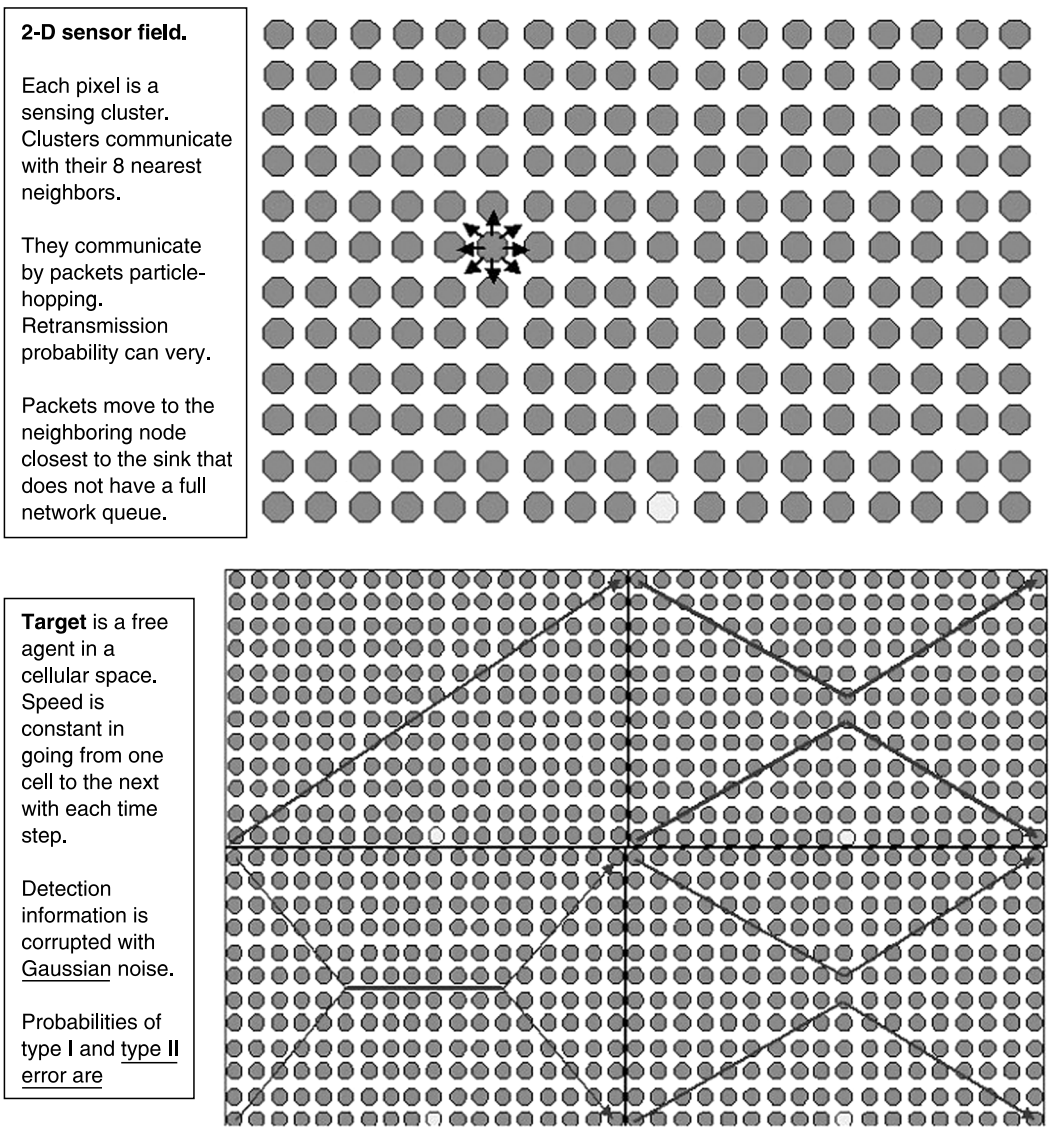


Figure 9.15. The top diagram explains the cellular automata model of the sensor field. The bottom diagrams show the four target trajectories used in our simulation scenarios.

© 2005 by Chapman & Hall/CRC

9.8 CA results

In this section, we present qualitative and quantitative results of CA simulations and a brief summary of the modeling techniques used. Our CA model is designed to mimic higher-level behaviors of clusters of sensor nodes. Each cell corresponds to a localized set of sensors nodes. A traffic sink is present. It connects our sensor grid to the outside world. Traffic is modeled at the cluster level. Simplifying assumptions facilitate the implementation of target tracking. We first present the results of target tracking algorithms, and follow them with network traffic analysis.

9.8.1 Linear Tracks

All three tracking algorithms adequately handle linear tracks. Figure 9.16(a) shows the track formed using pheromone tracking. This image shows the maximum pheromone concentrations achieved by cells in the sensor grid. At any point in time the pheromone concentrations are different since the abstract pheromones decay over time.

Figure 9.16(b) shows the same problem when EKF tracking is used. The fidelity of the model results in a near-constant covariance matrix P being computed at run time. The model uses constant cell sizes

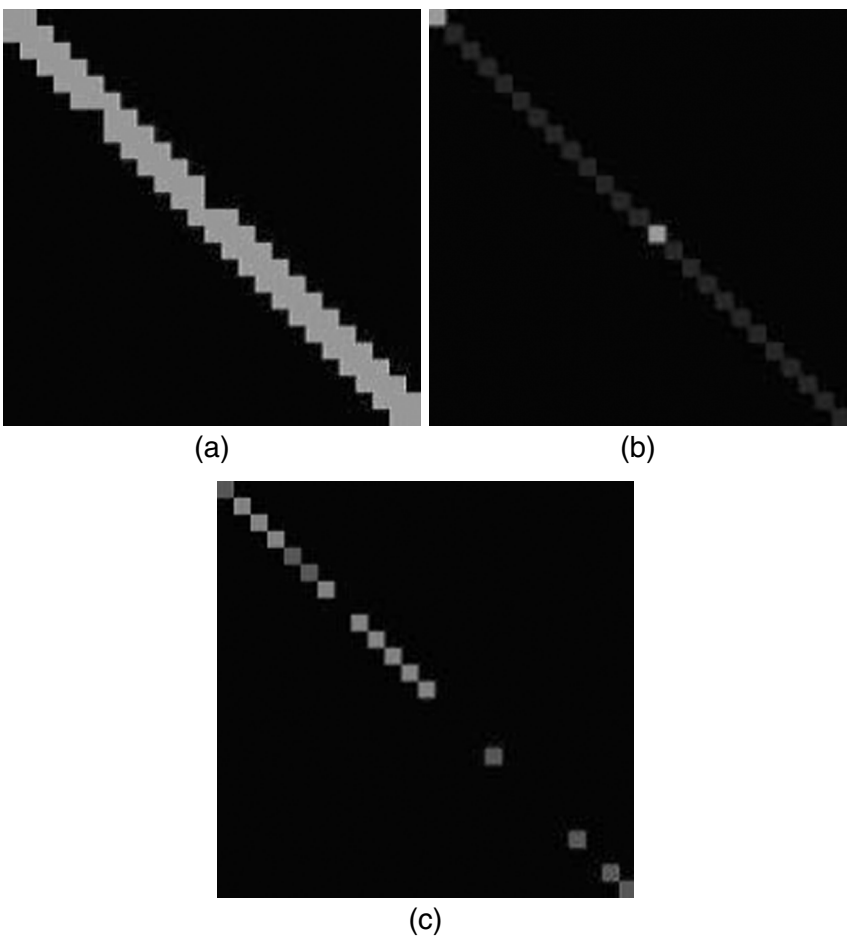


Figure 9.16. (a) The track created by the pheromone tracker when an entity crosses the terrain in a straight line from the upper left corner to the lower right corner. (b) Results of EKF tracking for the same scenario. (c) Results of the Bayesian belief network tracker.

© 2005 by Chapman & Hall/CRC

and Gaussian noise of uniform variance for the results of local collaboration. These simulations are used for a high-level comparison of algorithms and their associated resource consumption. The simplifications are appropriate for this. The expected value for vehicle position is the location of the cell receiving the detection. Speed is constant with Gaussian noise added. Gray cells indicate a track initiation; dark gray cells indicate a track continuation. In this case, an initiation occurs after the first observation of the agent. A second, erroneous initiation occurs later on as a result of noise.

Figure 9.16(c) presents the results from Bayesian net tracking. Gray squares show track initiation. Light gray squares indicate track continuation. Static conditional probabilities were used for each path through the net.

EKF tracking performs best when tracks are not ambiguous. The pheromone routing algorithm performs equally well; however, the track it constructs is significantly wider than the track produced by either the Bayesian net or EKF trackers. The track constructed by the Bayesian net algorithm contains gaps because of errors made in associating detections with tracks.

9.8.2 Crossing Tracks

When two tracks cross, track interpretation can be ambiguous. Unless vehicles can be classified into distinct classes, it is difficult to construct reliable tracks. Figure 9.17(a) demonstrates this using the pheromone tracker. Gray cells contain two vehicle pheromone trails contrasted to the dark and light gray cells that have only one vehicle. The track information is ambiguous when the vehicles deposit identical pheromones.

Figure 9.17(b) shows tracks formed using the EKF tracker. The target beginning in the lower left-hand corner was successfully tracked until it reached the crossing point. Here, the EKF algorithm was unable to identify the target successfully and began a new track shown by the gray cell. The second track was also followed successfully until the crossing point. After this point, the algorithm consistently propagated incorrect track information. This propagation is a result of the ambiguity in the track crossing. If an incorrect track is matched during disambiguation, then the error can be propagated forward for the rest of the scenario. In Figure 9.17(b), as in the other EKF images:

- Gray pixels signal track initiation.
- Dark gray pixels indicate correct track continuation.
- Light gray pixels signal incorrect track continuation.

In Figure 9.17(c) the central region of the sensor field using the Bayesian network approach image continued tracks in both directions. In these tests, we did not provide the Bayesian net with an appropriate disambiguation method. The network only knows that two vehicles passed, forming two crossing tracks. It did not estimate which vehicle went in which direction.

Each algorithm constructed tracks representing the shape of the vehicle path. Target-to-track matching suffered in the EKF, most likely as a result of model fidelity. The Bayesian net track-formation algorithm performed adequately and comparably to the pheromone-tracking model.

Pheromone tracking was able to construct both tracks successfully; additionally, different pheromone trails proved to be a powerful device in differentiating between vehicle tracks. Unfortunately, using different "digital pheromones" for each target type will differentiate crossed tracks only when the two vehicles are different. Creating a different pheromone for each track would currently be possible for different target classes when robust classification methods exist. Additional research is desirable to find applicable methods when this is not the case.

9.8.3 Nonlinear Crossing Tracks

We present three cases where the sensor network tracks nonlinear agent motion across the cellular grid. Figure 9.18(a) displays the pheromone trail of two vehicles following curved paths across the cellular grid. Contrast it with Figure 9.18(b) and (c), which shows EKF (Bayesian net) track formation. There is

© 2005 by Chapman & Hall/CRC

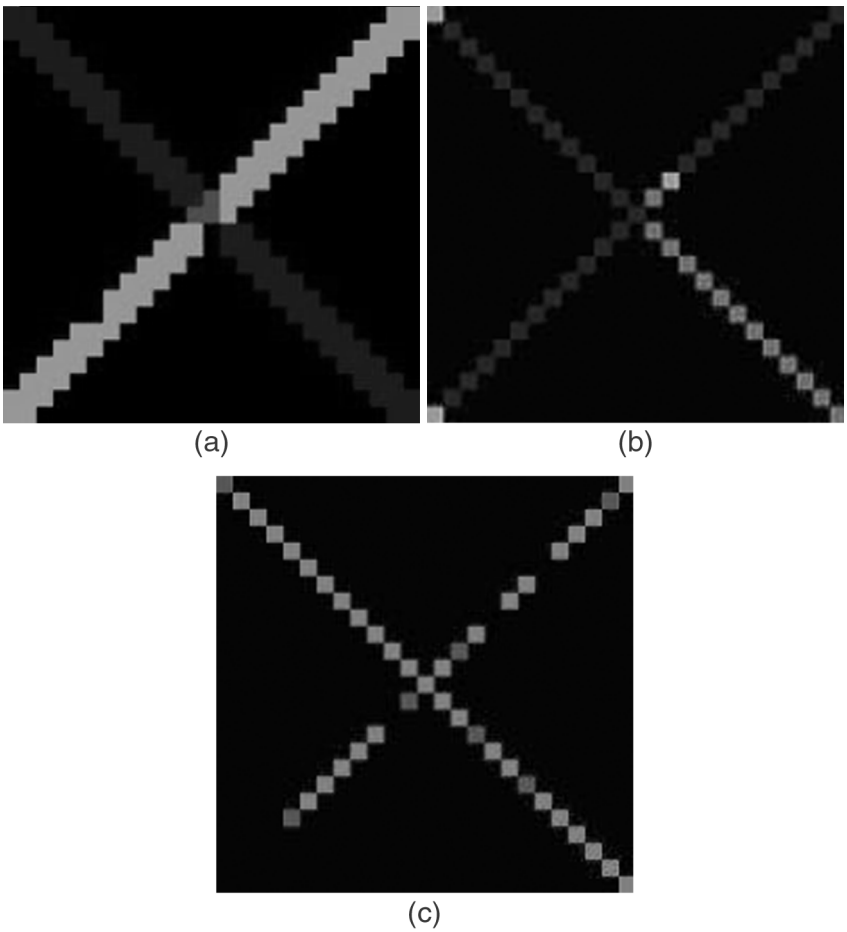


Figure 9.17. (a) Results when pheromones track crossing entities. (b) Results from EKF tracking of two crossing targets. (c) Bayesian belief net tracker applied to crossing tracks.

not much difference between the Bayesian net results for nonlinear tracks and that for linear crossing tracks. Pheromone results differ because two distinct pheromones are shown. If both vehicles had equivalent pheromones, then the two pheromone history plots would look identical. Pheromone concentration can indicate the potential presence of multiple entities. Figure 9.19 shows a plot of pheromone concentration over time around the track intersection. Regions containing multiple entities have higher pheromone concentration levels. Bayesian net formation, however, constructs a more crisp view of the target paths. Notice the ambiguity at the center, however. Here, we see a single-track continuation moving from bottom left to top right and two track initiations. These results indicate the inherent ambiguity of the problem.

9.8.4 Intersecting Tracks

Ambiguity increases when two tracks come together for a short time and then split. Figure 9.20(a) shows one such track formation. The middle section of the track would be ambiguous to the CA pheromone tracking algorithm if both vehicles are mapped to the same pheromone. Minor discontinuities occur in the individual tracks as a result of the agents' path through the cellular grid. The only information available is the existence of two tracks leaving a different pheromone. Figure 9.20(b) shows a plot of the pheromone levels through time. Clearly it is possible to use pheromone

© 2005 by Chapman & Hall/CRC

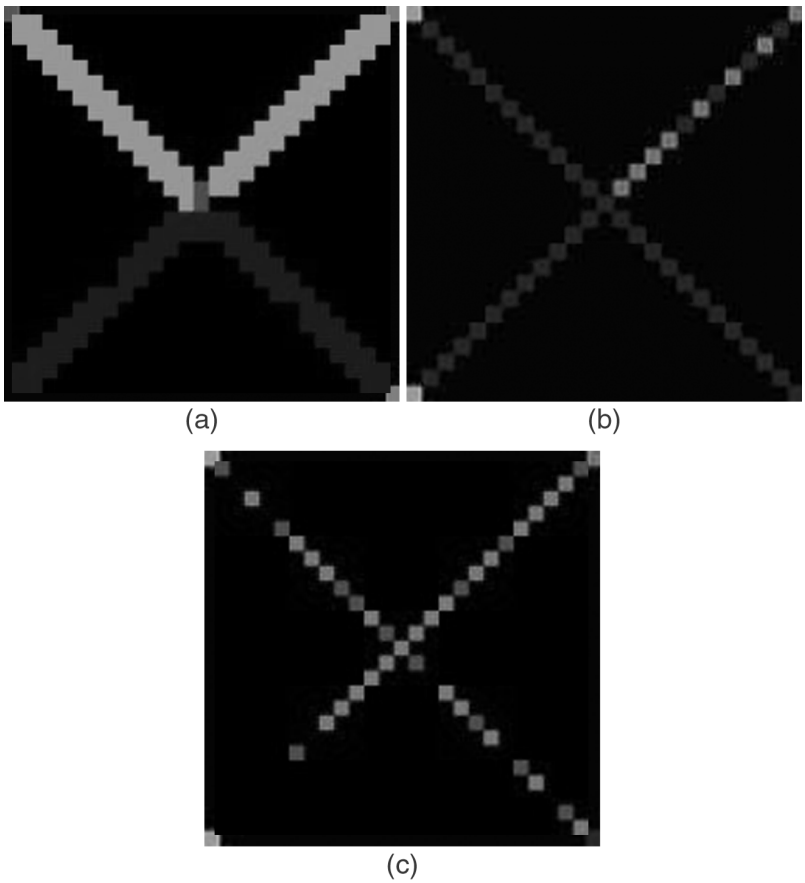


Figure 9.18. (a) Pheromone trails of two entities that follow nonlinear crossing paths. (b) EKF tracks of two entities in the same scenario. (c) Bayesian belief net tracks of crossing vehicles taking curved paths.

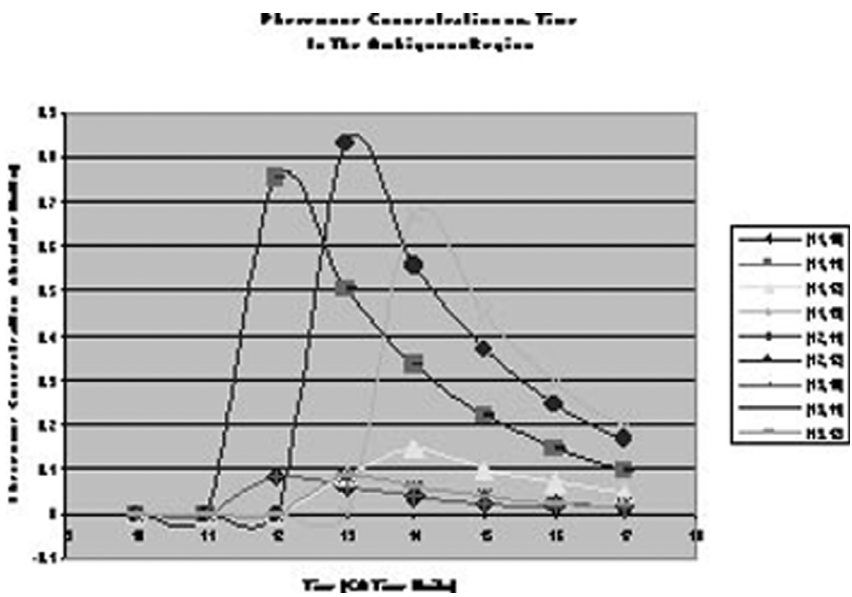


Figure 9.19. Pheromone concentration over time.

© 2005 by Chapman & Hall/CRC

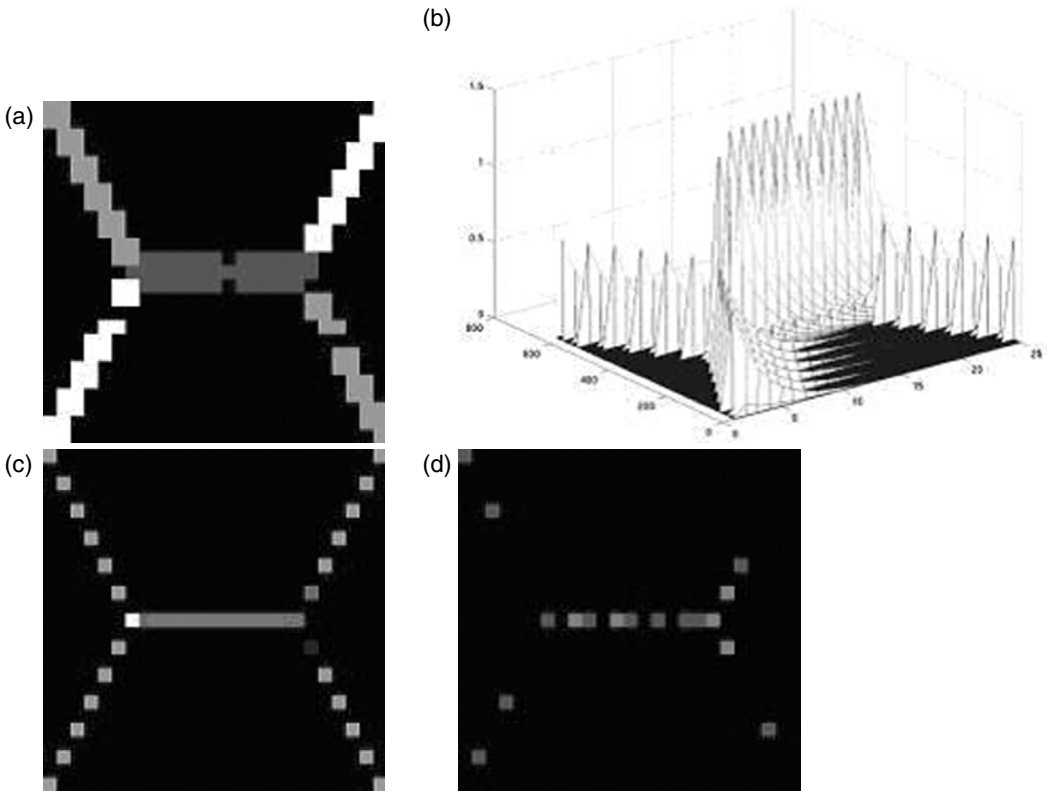


Figure 9.20. (a) The paths of two vehicles intersect, merge for a while and then diverge. In the absence of classification information, it is impossible to differentiate between two valid interpretations. (b) Pheromone concentrations produced in the scenario shown in (a). (c) EKF tracks formed in the same scenario as (a). (d) Results of Bayesian belief net tracker for this scenario.

concentration as a crude estimate for the number of collocated targets in a given region. Moreover, it may be possible to use the deteriorating nature of pheromone trails to construct a precise history for tracks in a given region.

In Figure 9.20(b), the central region in Figure 9.20(a) extends above the other peaks. This indicates the presence of multiple vehicles. Figure 9.20(c) shows the track produced by the EKF routing algorithm using the same agent path.

In our simulation, the ability of the EKF to manage this scenario depends on the size of the neighborhood for each cell. The curved path taken by the agent was imposed on a discrete grid. Doing so meant that detections did not always occur in contiguous cells. At this point it is not clear whether this error is a result of the low fidelity of the CA model or indicative of issues that will occur in the field. Our initial interpretation is that this error is significant and should be considered in determining sensor coverage. Ambiguity arises in the region where the tracks merge because both entities have nearly identical state vectors. Cells may choose one or the other with no deleterious affects on track formation. However, ground truth as it is represented in Figure 9.20(c) can only show that at least one of the two cells selected the incorrect track for continuation. This may also be a residual affect of the synchronous behaviors of the agents as they traverse the cellular grid.

Bayesian net track formation had the same problem with contiguous cells as the EKF tracker. Its performance was even more dependent on the ability of the system to provide continuous detections. If an agent activates two nonneighboring cells, then the probability of track continuation is zero, because no initial vehicle information was passed between the two nodes.

© 2005 by Chapman & Hall/CRC

9.8.5 Track Formation Effect on Network Traffic

Network traffic is a nonlinear phenomenon. Our model integrates network traffic analysis into the tracking algorithm simulations. This includes propagating traffic jams as a function of sensor network design. Figure 9.21 shows a sensor network randomly generating data packets. Traffic backups occur as data flows to the network sink. These simulations have data packets taking the shortest available route to the sink. When cell packet queues reach their maximum size, cells become unavailable. Packets detour around unavailable cells. Figure 9.22 shows the formation of a traffic jam. Figures 9.23(a) and (b) plots packet density in a region surrounding the sink. The legend indicates the (row–column) position of the cell generating the depicted queue-length history.

The existence and rate of growth of traffic jams around the sink is a function of the rate of information detection and the probability of successful data transmission. Consider false detections in the sensor grid, where p is the false alarm probability. For small p, no traffic jams form. If p increases beyond a threshold p_c, then traffic jams form around the sink. The value of p_c appears to be unique to each network. In our model it appears to be unique to each set of CA transition rules. This result is

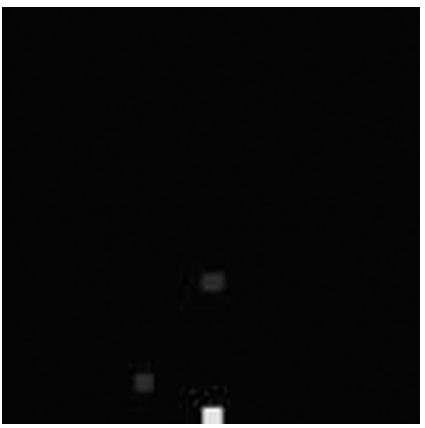


Figure 9.21. CA with random detections.

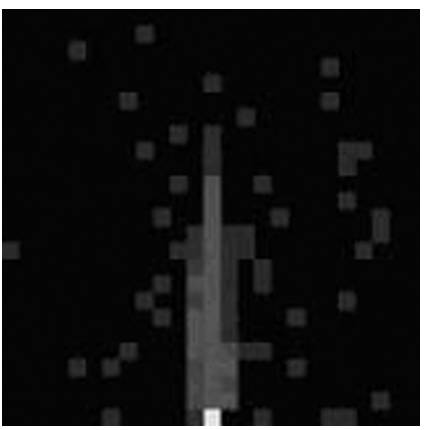


Figure 9.22. Traffic jam formation around the data sink. Data packets are generated randomly throughout the network. Light gray cells have maximum queue length. Black cells are empty. Darker shades of gray have shorter queue length than lighter shades.

© 2005 by Chapman & Hall/CRC

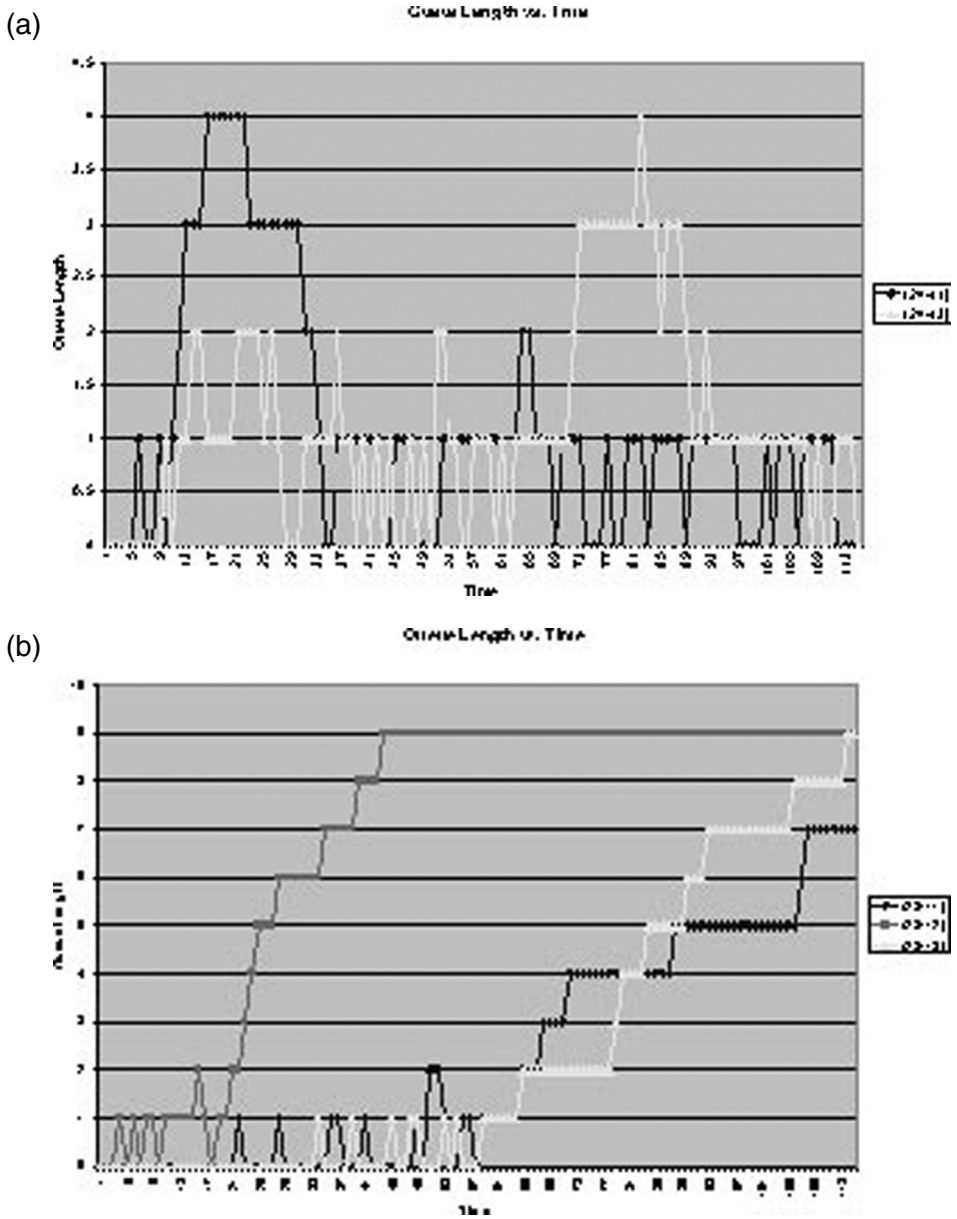


Figure 9.23. Average queue length versus time for nodes surrounding the data sink when probability of false alarm is (a) below and (b) above the critical value.

consistent with queuing theory analysis, where maximum queue length tends to infinity when the volume of requests for service is greater than the system's capacity to process requests.

When detections occur, data packets are passed to neighboring cells in the direction the entity is traveling. Neighboring cells store packets and use this data for track formation. Packets are also sent to the data sink along the shortest path. This simple routing algorithm causes traffic jams to form around the network sink. A vertical path above the sink forms, causing a small traffic jam.

The image in Figure 9.24(c) shows the queue length of the column in 54 steps. The first ten rows of data have been discarded. The remaining rows illustrate the traffic jam seen in Figure 9.24(b).

© 2005 by Chapman & Hall/CRC

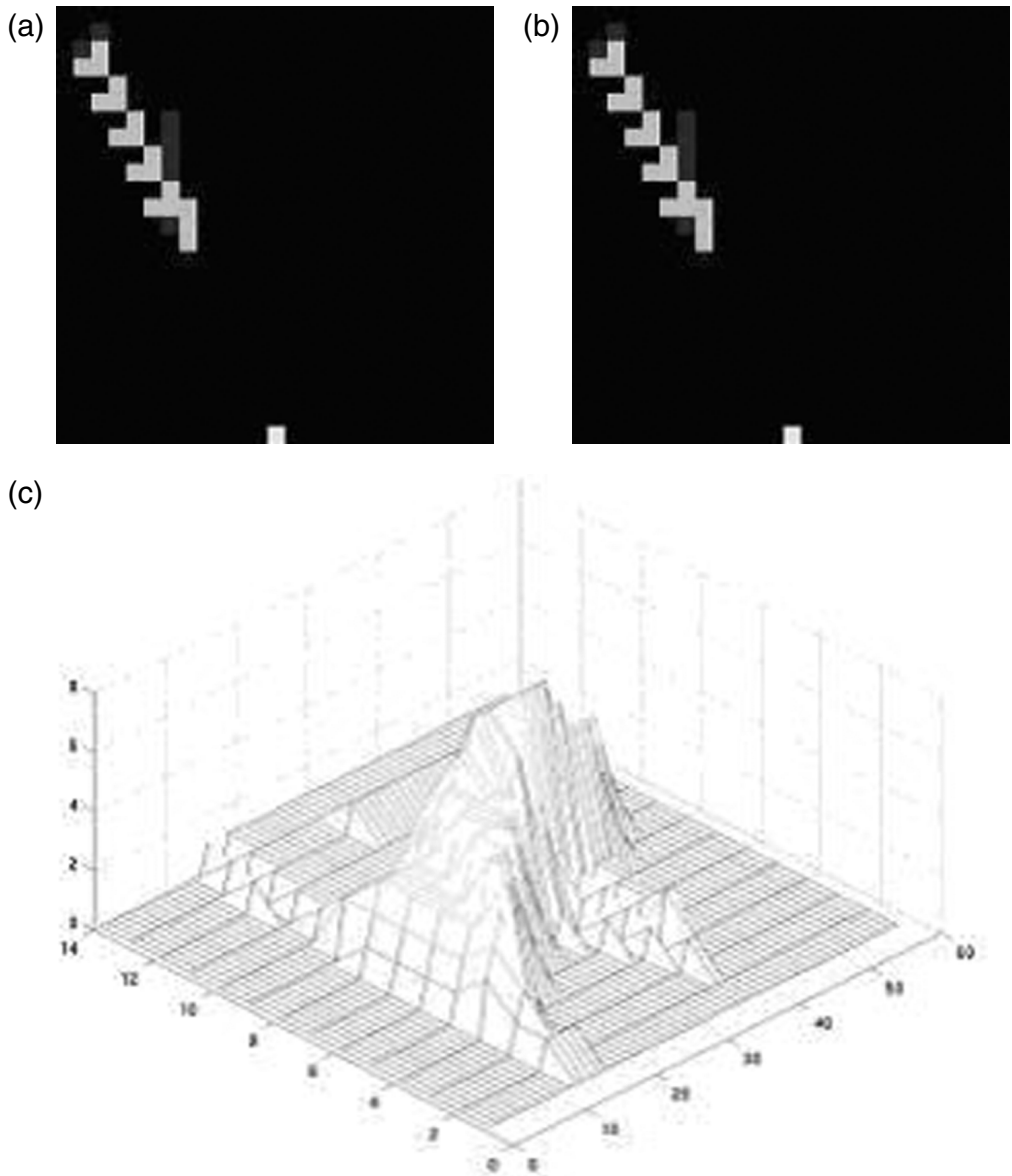


Figure 9.24. (a) Data packet propagation in the target-tracking scenario shown in Figure 9.16. (b) Formation of a traffic jam above the data sink. (c) Three-dimensional view of traffic jam formation in the 12th column of the CA grid.

Traffic flux through the sink is proportional to the number of tracks being monitored, to the probability of false detection, and to the probability of a successful transmission. Assuming a perfect network and nonambiguous tracks, this relationship is linear, e.g.

$$\varphi_{\text{sin k}} = kT$$

Where T is the number of tracks and φ is the flux. Track ambiguities and networking imperfections cause deviations from this linear structure. The exact nature of the distortion depends directly on the CA transition rule and the type of track uncertainty.

© 2005 by Chapman & Hall/CRC

9.8.6 Effects of Network Pathologies

Sensing and communications are imperfect in the real world. In this section we analyze the effects of imperfections on track formation. We will analyze the case where false positives occur with a probability of 0.001 per cell per time step. Figure 9.25(a) shows a pheromone track constructed in the presence of false positives:

Figure 9.25(b) illustrates how the existence of false positives degrades the performance of the pheromone tracker. The first peak is false detection. The peak just below this shows the spread of the pheromone. Both peaks decay until an actual detection is made. The first peak could be interpreted as the beginning of a track. This misinterpretation is minor in this instance, but these errors could be significant in other examples.

It may be possible to use pheromone decay to modify track continuation probabilities in these cases. If a pheromone has decayed beyond a certain point, then it could be assumed that no track was created. In the example, the false detection decayed below a concentration of 0.2 pheromone units before the jump due to the actual sensor detection. If 0.2 were the cut off for track continuation, then the node located at cell grid (8, 4) would have constructed a track continuation of the true track, not the false one. Further studies would help us determine the proper rates for pheromone diffusion and decay.

Figure 9.25(c) shows the track formed using the EKF approach in the presence of false positives. The algorithm is relatively robust to these types of error. However, as was shown in Figure 9.20(c), lack of contiguous target sightings plays a significant role in degrading track fidelity.

Of the three algorithms studied, the pheromone approach formation is most robust to the presence of false positives. The decay of pheromones over time allows the network to isolate local errors in space

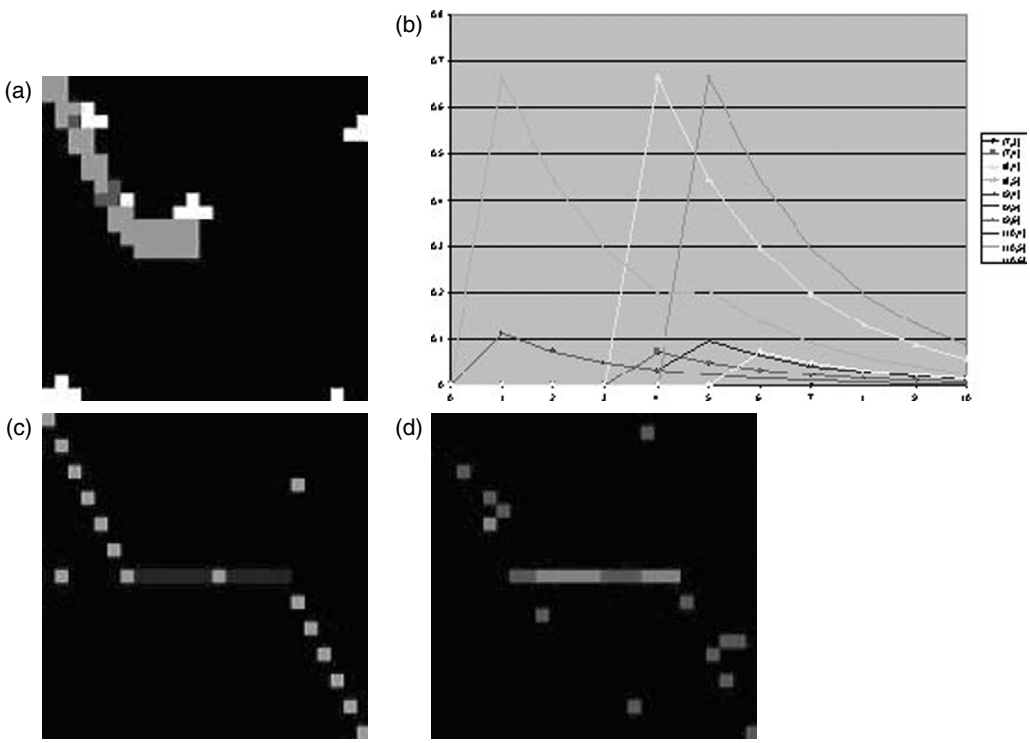


Figure 9.25. (a) Pheromone track formed with false positive inputs. White areas indicate false alarms; darker gray areas indicate regions where two vehicles were detected. (b) Ambiguity in pheromone quantities when a track crosses an area with a false positive. (c) The EKF filter tolerates false positives, but is sensitive to the lack of continuous sensor coverage. (d) Bayesian net tracking in the presence of false positives.

© 2005 by Chapman & Hall/CRC

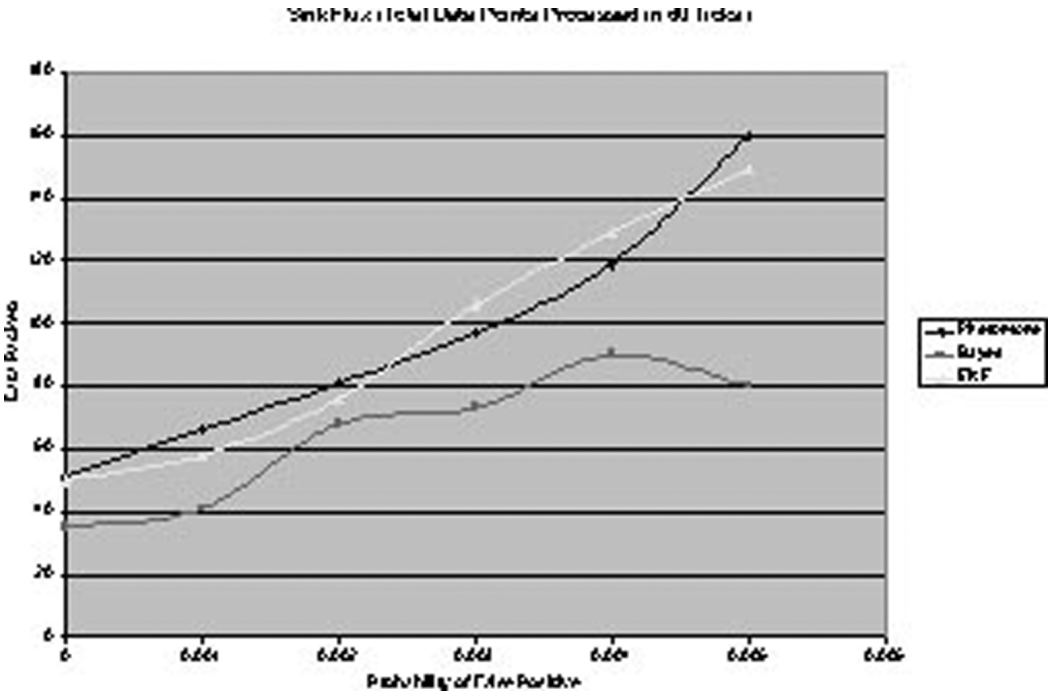


Figure 9.26. Probability of a false positive versus the volume of data flowing through the data sink.

and time. The confidence level of information in pheromone systems is proportional to the concentration of the pheromone itself. Thus, as pheromones diffuse through the grid, their concentration, and thus the level of confidence, decreases. Once the pheromone concentration drops below a certain threshold its value is truncated to zero. EKF and Bayesian net track formation is dependent on the spread of information to create track continuations. If a false detection is made near an existing track, then it causes errors to occur in initial track formation; which then propagates throughout the network.

Network traffic is also affected by the existence of false positives. All detections are transmitted to the central sink for processing. Track formation information is also transmitted to surrounding nodes. As suggested by our empirical analysis, traffic jams form around the sink when the false positive probability is higher than 0.002 for this particular CA. This is aggravated by the propagation of false positives. Figure 9.26 displays the relationship between false positive detection and flux through the sink.

The Bayesian net generates fewer data packets than the other two. The algorithm is designed to disregard some positive readings as false. The others do not. The EKF assumes Gaussian noise. Pheromones propagate uncertain data to be reinforced by others.

Imperfect data transmission also affects sensor networks. Figure 9.27(a)–(c) displays the tracks-formed by the three track-formation algorithms when the probability of inter-cell communication is reduced to 75%. This means that packets frequently need to be retransmitted.

Tracking performance is not badly affected by the lack of timely information. The pheromone track is the least affected by the transmission loss. The fidelity of the track is slightly worse than observed in a perfect network. The track formed by the EKF algorithm is more severely affected. When information is not passed adequately, track continuation is not possible. This leads to a number of track initiations. A similar effect is noted in the Bayesian net track. It is clear that pheromone track formation is the most resilient to the lack of punctual data because the track does not rely on the sharing of information for track continuation. The other two algorithms rely on information from neighbors to continue tracks.

© 2005 by Chapman & Hall/CRC

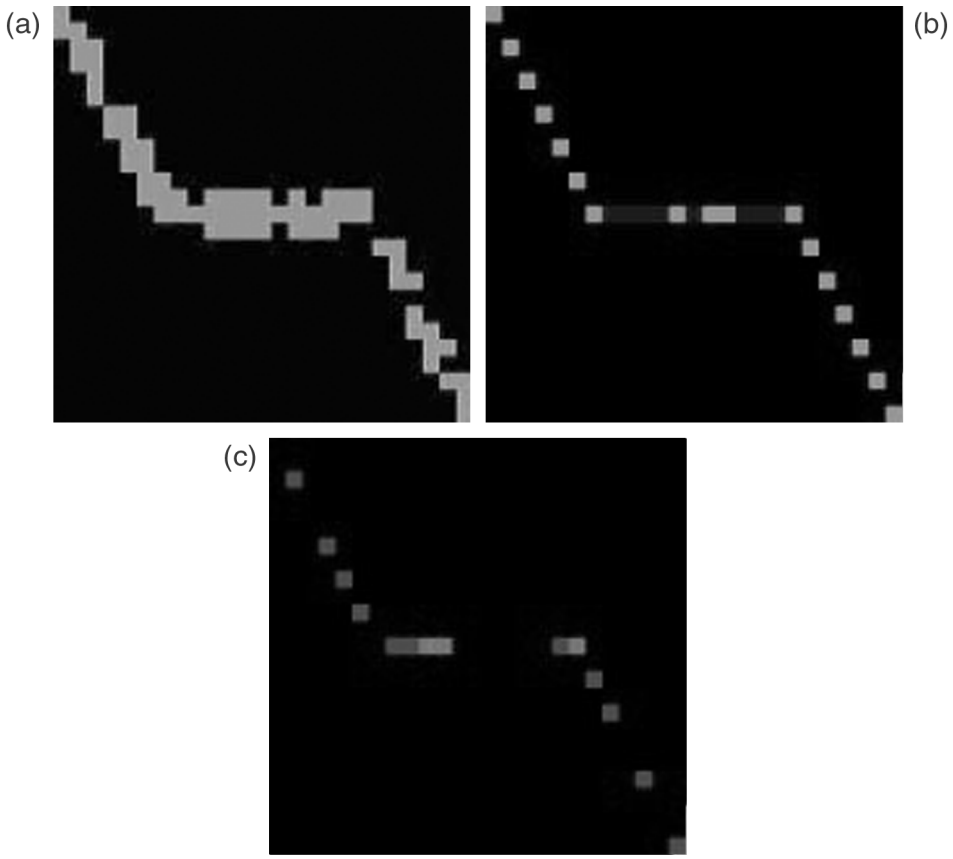


Figure 9.27. (a) Pheromone track formed in the presence of frequent data retransmissions. (b) EKF track in the same conditions. (c) Bayesian track formed in an imperfect network.

9.9 Collaborative Tracking Network

The Collaborative tracking network (ColTraNe) is a fully distributed target tracking system. It is a prototype implementation of the theoretical inter-cluster distributed tracker presented above. ColTraNe was implemented and tested as part of a larger program. Sensoria Corporation constructed the sensor nodes used. Individual nodes use SH4 processors running Linux and are battery powered. Wireless communication for ColTraNe uses time division multiplexing. Data routing is done via the diffusion routing approach [14], which supports communications based on data attributes instead of node network addresses. Communications can be directed to geographic locations or regions.

Each node had three sensor inputs: acoustic, seismic, and passive infrared (PIR). Acoustic and seismic sensors are omni-directional and return time-series data. The PIR sensor is a two-pixel imager. It detects motion and is directional. Software provided by BAE Systems in Austin, Texas, handles target detection. The software detects and returns CPA events. CPA is a robust, easily detected statistic. A CPA event occurs when the signal intensity received by a sensor starts to decrease. Using CPA events from all sensor types makes combining information from different sensing modes easy. Combining sensory modes makes the system less affected by many types of environmental noise [5].

We summarize the specific application of Figure 9.2 to ColTraNe as follows:

1. Each node waits for CPA events to be triggered by one or more of its sensors. The node also continuously receives information about target tracks heading towards it.

© 2005 by Chapman & Hall/CRC

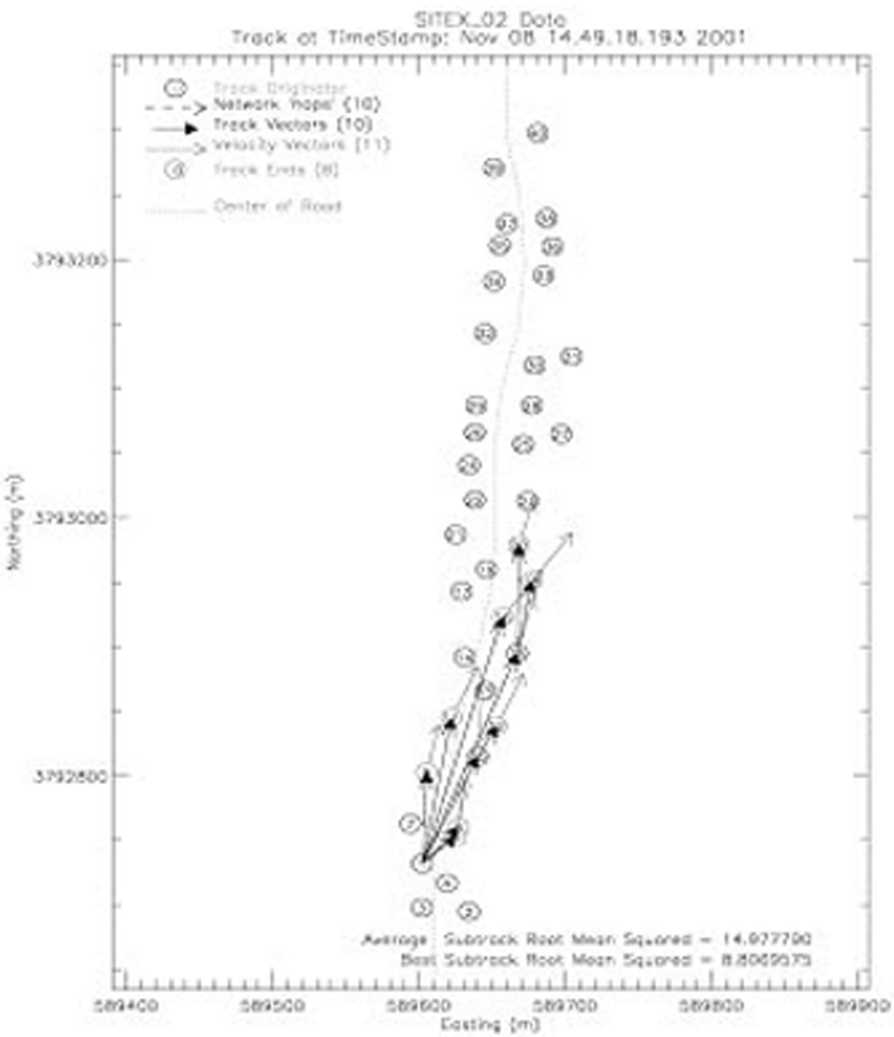


Figure 9.28. Tracks from a sample target tracking run at Twenty Nine Palms. Both axes are UTM coordinates. Circles are sensor nodes. The faint curve through the nodes is the middle of the road. Dark arrows are the reported target tracks. Dotted arrows connect the *clump heads* that formed the tracks. Filtering not only reduced the system's tendency to branch, but also increased the track length. (a) No filtering. (b) 45-Degree Angle Filter. (c) Extended Kalman Filter. (d) Lateral Inhibition.

2. When a CPA event occurs, relevant information (node position, CPA time, target class, signal intensity, etc.) is broadcast to nodes in the immediate vicinity.
3. The node with the most intense signal in its immediate neighborhood and current time slice is chosen as the local *clump head*. The clump head calculates the geometric centroid of the contributing nodes' positions, weighted by signal strength. This estimates the target position. Linear regression is used to determine target heading and velocity.
4. The clump head attempts to fit the information from step 3 to the track information received in step 1. We currently use a Euclidean metric for this comparison.
5. If the smallest such track fit is too large, or no incoming track information is found, then a new track record is generated with the data from step 3. Otherwise, the current information from step 3 is combined with the information from the track record with the closest track fit to create an updated track record.

© 2005 by Chapman & Hall/CRC

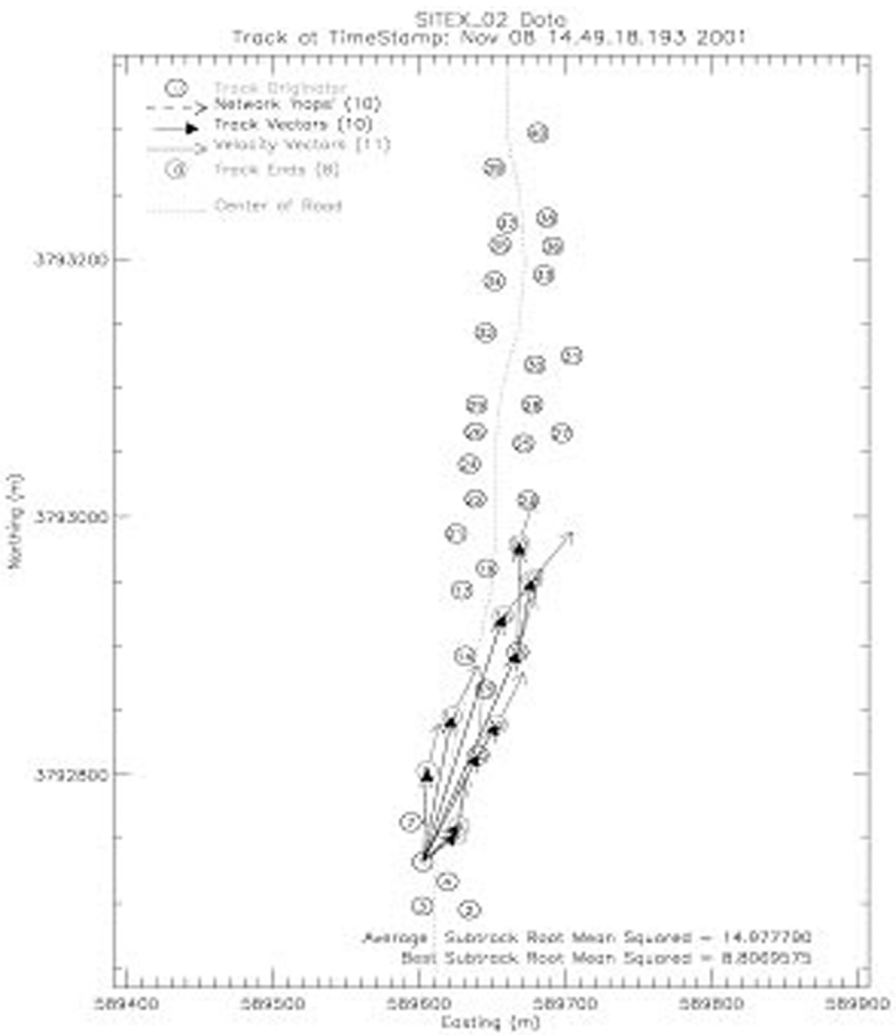


Figure 9.28. Continued.

6. The record from step 5 is transmitted to the user community.
7. A region is defined containing the likely trajectories of the target; the track record from step 5 is transmitted to all nodes within that region.

Of the three track-maintenance techniques evaluated above (pheromone, EKF, and Bayesian), field tests showed that the EKF concept was feasible. Processing and networking latency were minimal and allowed the system to track targets in real time.

Distributing logic throughout the network had unexpected advantages in our field test at Twenty Nine Palms in November 2001. During the test, hardware and environmental conditions caused 55% of the CPA events to be false positives. The tracks initiated by erroneous CPA events were determined by step 3 to have target heading and velocity of 0.0, thereby preventing their propagation to the rest of the nodes in step 7. Thus, ColTraNe automatically filtered this clutter from the data presented to the user.

Problems with the Twenty Nine Palms implementation were also discovered:

1. The implementation schedule did not allow the EKF version of step 5 to be tested.
2. The velocity estimation worked well, but the position estimation relied on the position of the clump head.

© 2005 by Chapman & Hall/CRC

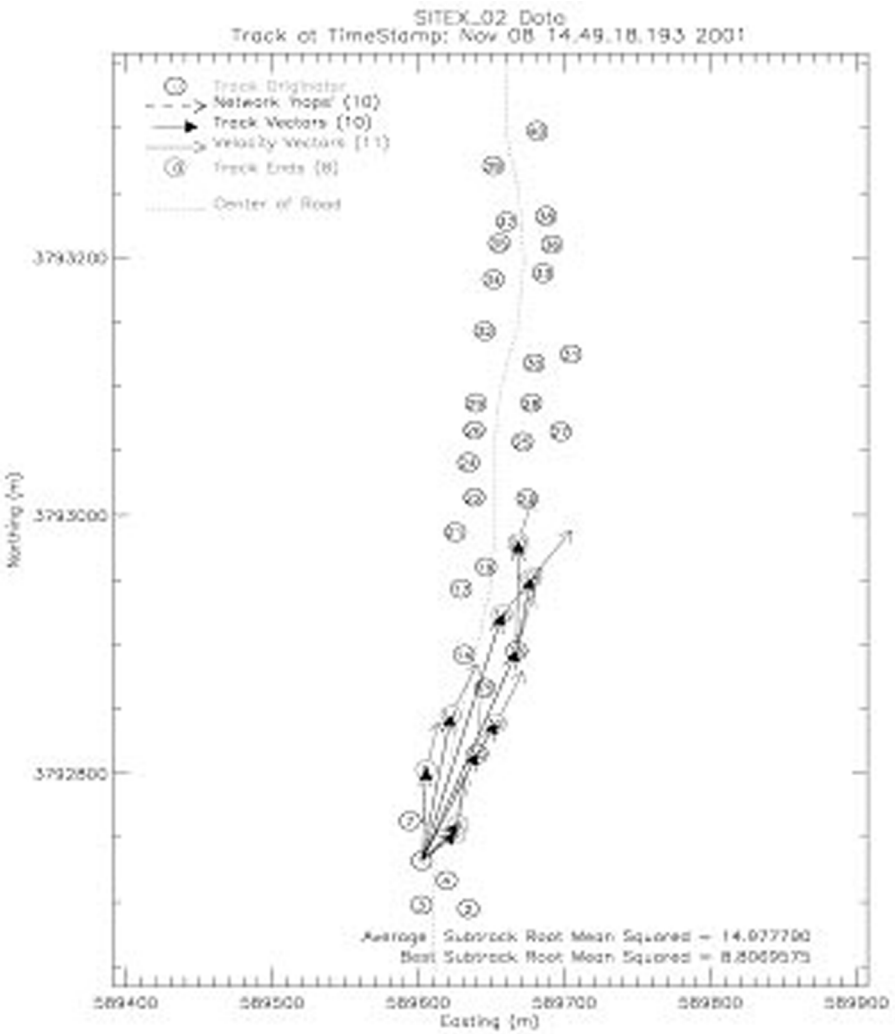


Figure 9.28. Continued.

3. The tracks tended to branch, making the results difficult to decipher (see Figure 9.28(a)).
4. The tracking was limited to one target at a time.

Continued development has alleviated these problems. The EKF Filter was integrated into the system. This improved the quality of both track and target position estimates as tracks progress.

An angle gate, which automatically excludes continuations of tracks when velocity estimates show targets are moving in radically different directions, has been inserted into the track-matching metric. This reduces the tendency of tracks to branch, as shown in Figure 9.28(b).

We constructed a technique for reducing the tendency of tracks to branch. We call this technique lateral inhibition. Before continuing a track, nodes whose current readings match a candidate track broadcast their intention to continue the track. They then wait for a period of time proportional to the log of their goodness-of-fit value. During this time, they can receive messages from other nodes that fit the candidate track better. If better fits are received, then they drop their continuations. If no one else reports a better fit within the timeout period, then the node continues the track. Figure 9.28(d) shows a target track with lateral inhibition.

© 2005 by Chapman & Hall/CRC

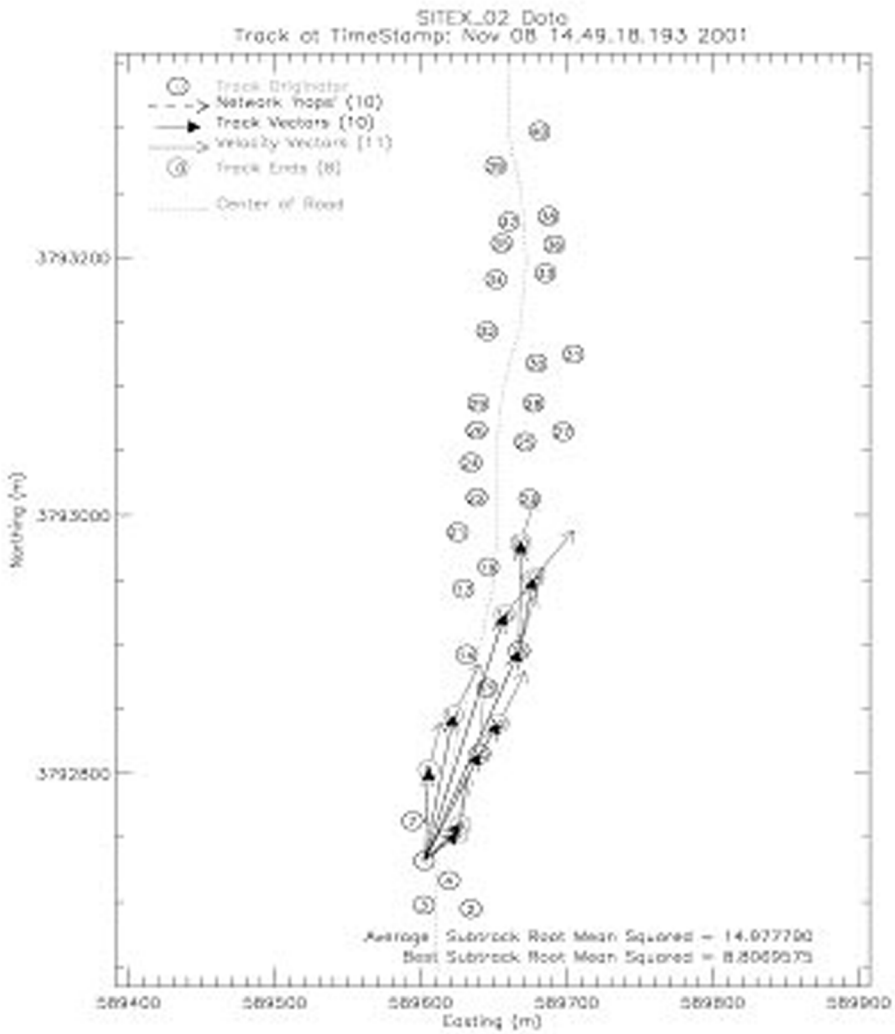


Figure 9.28. Continued.

The target position is now estimated as the geometric centroid of local target detections with signal intensity used as the weight. Our tests indicate that this is more effective in improving the position estimate than the EKF. The geometric centroid approach was used in the angle filter and lateral inhibition test runs shown in Figure 9.28 and Table 9.1.

Differences between the techniques can be seen in the tracks in Figure 9.28. The tracks all use data from a field test with military vehicles at Twenty Nine Palms Marine Training Ground. Sensor nodes were placed along a road and at an intersection. In the test run depicted in Figure 9.28, the vehicle traversed the sensor field along a road going from the bottom of the diagram to the top. The faint dotted line shows the position of the center of the road. Figure 9.28(a) shows the results from our original implementation. The other diagrams use our improved techniques and the same sensor data.

Figure 9.28(a) illustrates the deficiencies of our original approach. The tracking process works, but many track branches form and the results are difficult to interpret. Introducing a 45° angle gate (Figure 9.28(b)) reduces track branching. It also helps the system correctly continue the track further than our original approach. Estimating the target position by using the geometric centroid greatly

© 2005 by Chapman & Hall/CRC

Table 9.1. Root-mean-square error comparison for the data association techniques discussed. The top set of numbers is for all target tracks collected on the day of November 8. The bottom set of numbers is for one specific target run. In each set, the top row is the average error for all tracks made by the target during the run. The bottom row sums the error over the tracks. Since these tests were of a target following a road, the angle and EKF filters have an advantage. They assume a linear trajectory. Lateral inhibition still performs well, although it is non-parametric

	Live data	Angle 45°	EKF	Lateral inhibition	EKF & Lat
RMS for tracks from Nov 08 2001					
Averaged	18.108328	9.533245	8.877021	9.361643	11.306236
Track summed	81.456893	54.527057	52.775338	13.535534	26.738410
RMS for track beginning at Nov_08_14.49.18.193_2001					
Averaged	14.977790	8.818092	8.723196	9.361643	8.979458
Track summed	119.822320	123.453290	183.187110	18.723287	35.917832

improves the accuracy of the track. This approach works well because it assumes that targets turn slowly, and in this test the road section is nearly straight.

Using the EKF (Figure 9.28(c)) also provides a more accurate and understandable set of tracks. Branching still occurs, but it is limited to a region that is very close to the actual trajectory of the target. The EKF performs its own computation of the target position. Like the angle filter, the EKF imposes a linear model on the data, and hence works well with the data from the straight road.

The lateral inhibition results (Figure 9.28(d)) have the least amount of track branching. This track is the most easily understood of all the methods shown. It is nonparametric and does not assume linearity in the data. As with the angle gate, the geometric centroid is a good estimate of the target position.

We have also tested a combination of EKF and lateral inhibition. The results of that approach are worse than either the EKF or lateral inhibition approaches in isolation.

Our discussion of the track data is supported by the error data summarized in Table 9.1. Each cell shows the area between the track formed by the approach and the actual target trajectory. The top portion of the table is data from all the tracks taken on November 8, 2001. The bottom portion of the table is from the track shown in Figure 9.28. In both portions, the top row is the average error for all the tracks formed by a target. The bottom row is the sum of all the errors for all the tracks formed by a target.

If one considers only the average track error, the EKF provides the best results. The original approach provides the worst results. The other three approaches considered are roughly equivalent.

Summing the error of all the tracks formed for a single target penalizes approaches where multiple track branches form. When this is done, lateral inhibition has the most promising results. The second best results are provided by the combination of lateral inhibition and EKF. The other approaches are roughly equivalent.

These results show that the inter-node coordination provided by lateral inhibition is a promising technique. Since it is nonparametric it makes very few assumptions about the target trajectory. Geometric centroid is a robust position estimator. Robust local parameter estimation provides a reliable estimate of the target's position and heading. Lateral inhibition reduces the tendency of our original tracking implementation to produce confusing interpretations of the data inputs. The system finds the track continuation that is the best continuation of the last known target position. In combination, both methods track targets moving through a sensor field more clearly.

The distributed nature of this approach makes it very robust to node failure. It also makes multiple target-tracking problems easy to solve when targets are much more sparsely distributed than the sensors. Multiple target tracking becomes a disjoint set of single-target tracking problems.

Multiple-target tracking conflicts arise only when target trajectories cross each other or approach each other too closely. When tracks cross or approach each other closely, linear regression breaks down

© 2005 by Chapman & Hall/CRC

Table 9.2. Data transmission requirements for the different data association techniques. The total is the number of bytes sent over the network. The EKF requires covariance data and previous data points. Angle gating and lateral inhibition require less data in the track record. Data are from the tracking period shown in Figure 9.28

	Track packets	Track pack size	CPA packets	Inhibition/packets	Total
EKF	852	296	59	0	254552
Lateral inhibition	217	56	59	130	21792
EKF & lateral inhibition	204	296	59	114	69128
Centralized	0	0	240	0	9600

CPA size 40; inhibition size 56.

since CPA events from multiple targets will be used in the same computation. The results will tend to match neither track. The tracks will be continued once the targets no longer interfere with each other.

Classification algorithms will be useful for tracking closely spaced targets. If crossing targets are of different classes and class information is transmitted as part of the CPA event, then the linear regression could be done on events grouped by target class. In which case, target crossing becomes even less of a concern.

Table 9.2 compares the network traffic incurred by the approaches shown in Figure 9.28 with the bandwidth required for a centralized approach using CPA data. CPA packets had 40 bytes, and the lateral inhibition packets had 56 bytes. Track data packets vary in size, since the EKF required three data points and a covariance matrix. Table 9.2 shows that lateral inhibition requires the least network bandwidth due to reduced track divergence.

Note from Table 9.2, that in this case centralized tracking required less than half as many bytes as lateral inhibition. These data are somewhat misleading. The data shown are from a network of 40 nodes with an Internet gateway in the middle. As the number of nodes and the distance to the gateway increases, so the number of packet transmissions will increase for the centralized case. For the other techniques, the number of packets transmitted will remain constant. Recall the occurrence of tracking filter false positives in the network, which was more than 50% of the CPAs during this test. Reasonably, under those conditions the centralized data volume would more than double over time and be comparable to the lateral inhibition volume. Note also that centralized data association would involve as many as 24 to 30 CPAs for every detection event in our method. When association requires $O(n^2)$ comparisons [15] this becomes an issue.

9.10 Dependability Analysis

Our technology allows the clump head that combines readings to be chosen on the fly. This significantly increases system robustness by allowing the system to adapt to the failure of individual nodes. The nodes that remain can exchange readings and find answers.

Since our heading and velocity estimation approach uses triangulation [2], at least three sensor readings are needed to get an answer. In the following, we assume that all nodes have an equal probability of failure q. In a nonadaptive system, when the cluster head fails the system fails. The cluster has a probability of failure q no matter how many nodes are in the cluster. In the adaptive case, the system fails only when the number of nodes functioning is three or less.

Figures 9.29 and 9.30 illustrate the difference in dependability between adaptive and nonadaptive tasking. These figures assume an exponential distribution of independent failure events, which is standard in dependability literature. The probability of failure is constant across time. We assume that all participating nodes have the same probability of failure. This does not account for errors due to loss of power.

© 2005 by Chapman & Hall/CRC

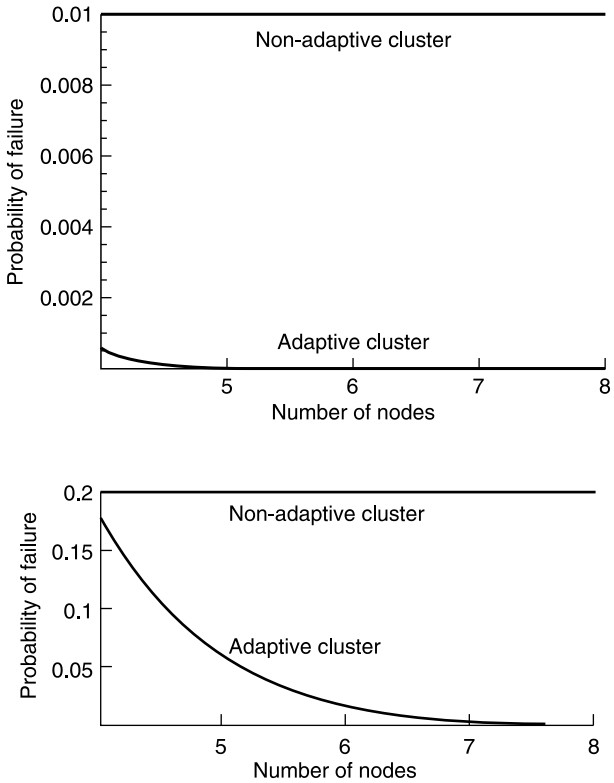


Figure 9.29. Probability of failure q: (a) 0.01; (b) 0.02. The number of nodes in the cluster is varied from four to eight.

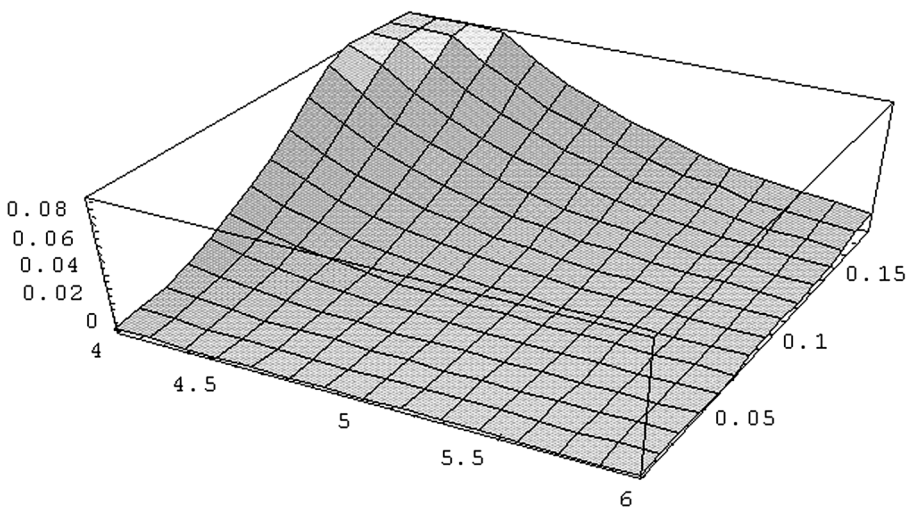


Figure 9.30. The surface shows probability of failure (z axis) for an adaptive cluster as the probability of failure for a single node q varies from 0.01 to 0.2 (side axis), and the number of nodes in the cluster varies from four to six (front axis).

© 2005 by Chapman & Hall/CRC

In Figure 9.29 the top line is the probability of failure for a nonadaptive cluster. Since one node is the designated cluster head, when it fails the cluster fails. By definition, this probability of failure is constant. The lower line is the probability of failure of an adaptive cluster as a function of the number of nodes. This is the probability that less than three nodes will be available at any point in time. All individual nodes have the same failure probability, which is the value shown by the top line. The probability of failure of the adaptive cluster drops off exponentially with the number of nodes.

Figure 9.30 shows this same probability of failure as a function of both the number of nodes and the individual node's probability of failure.

9.11 Resource Parsimony

We also constructed simulations to analyze the performance and resource consumption of ColTraNe. It is compared with a beamforming algorithm from Yao *et al.* [16]. Each approach was used on the same set of target tracks with the same sensor configurations. Simulated target tracks were constructed according to

$$y_t = \alpha(x+4) + (1-\alpha)(x^2/4)$$

The sensors were arrayed in a grid over the square area of $x \in [-4, 4]$ and $y \in [0, 8]$. Four configurations consisting of 16, 36, 64, and 100 sensors were constructed in order to examine the effects of sensor density on the results. For each density, five simulations were run for $\alpha = 0$, 0.25, 0.5, 0.75, and 1, each of which relied on simulated time-series data (in the case of beamforming) and simulated detections (in the case of the CPA-based method).

Parameters measured included average error rate, execution time, bandwidth consumed, and power (or more properly, energy) consumed. Average error is a measure of how much the average estimated target position deviated from the true target position. Bandwidth consumption corresponded to the total amount of data exchanged over all sensor nodes throughout the lifetime of the target track. Power consumption was measured taking into account both the power required by the CPU for computation and the power required by the network to transmit the data to another node.

The resulting graphs are displayed in Figures 9.31 and 9.32. The results for beamforming in Figure 9.31 show that it is possible to reduce power consumption considerably without significantly influencing average error. In the case of ColTraNe, the lowest error resulted from expending power somewhere between the highest and lowest amounts of consumption. Comparing the two algorithms, beamforming produced better results on average, but consumed from 100 to 1000 times as much power as the CPA-based method, depending on the density of the sensor network.

9.12 Multiple Target Tracking

To analyze the ability of ColTraNe to track multiple targets we performed the following experiment. Two simulated targets were sent through a simulated sensor node field comprised of 400 nodes arranged in a rectangular grid measuring $8 \times 8\,\mathrm{m}^2$. Two different scenarios were used for this simulation:

1. X path. Two targets enter the field at the upper and lower left corners, traverse the field, crossing each other in the center of the field, and exit at the opposite corners. See Figure 9.33(a).
2. Bowtie. Two targets enter the field at the upper and lower left corner, traverse the field along hyperbolic paths that nearly intersect in the center of the field, and then exit the field at the upper and lower right corners. See Figure 9.33(b).

Calculation of the tracking errors was accomplished by determining the area under the curve between a track plot and the target path to which it was related.

© 2005 by Chapman & Hall/CRC

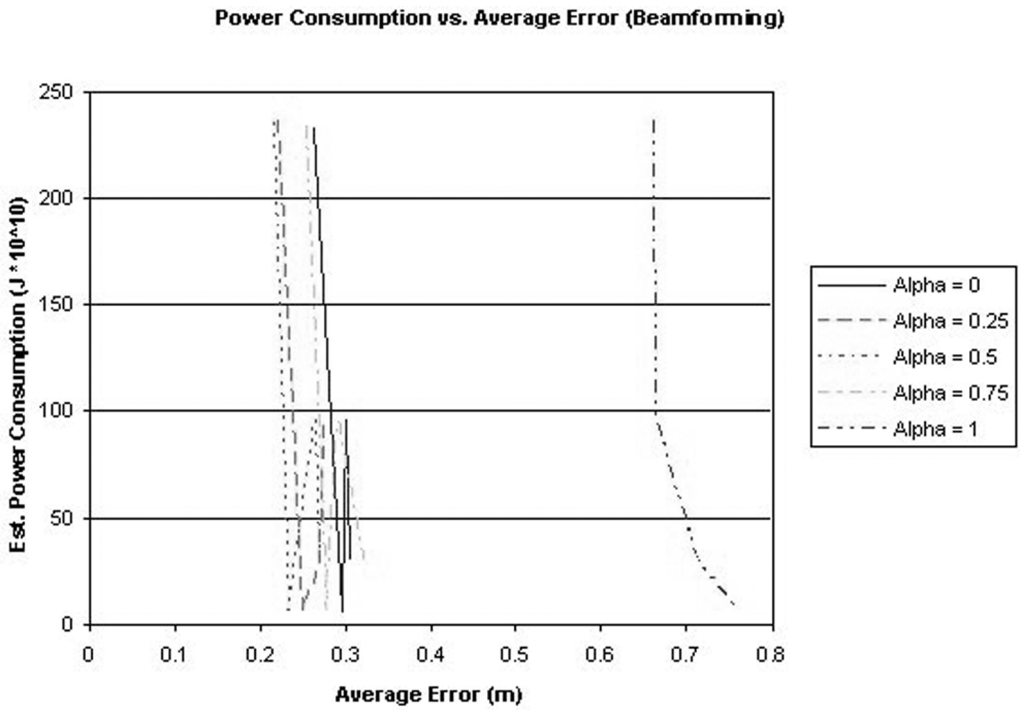


Figure 9.31. Beamforming, power consumption vs. average error.

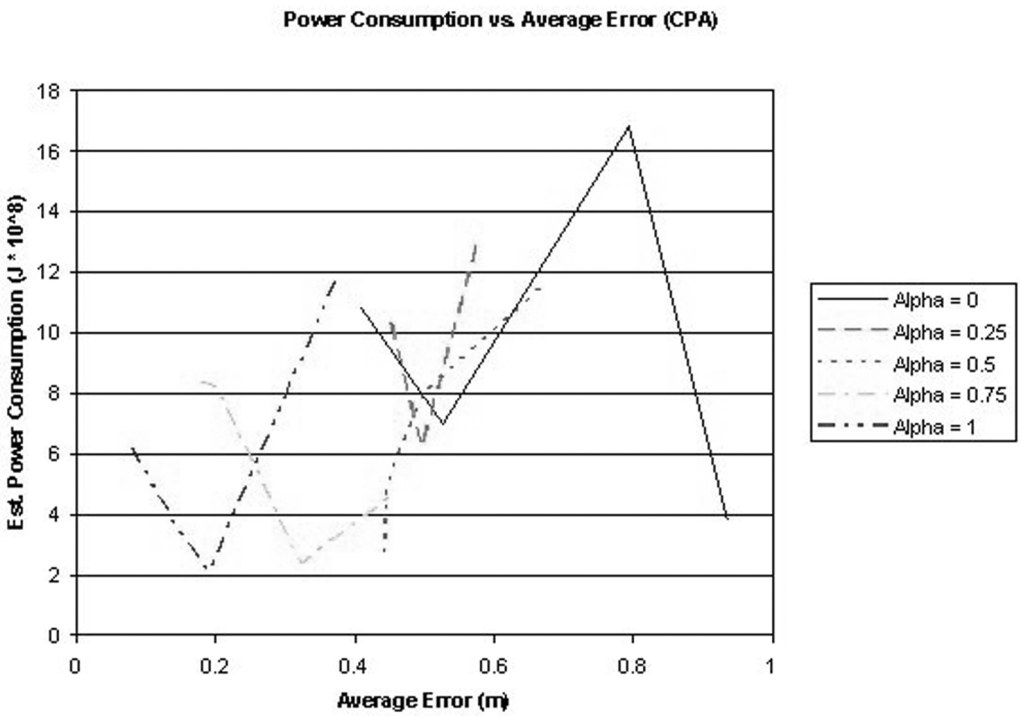


Figure 9.32. CPA, power consumption vs. average error.

© 2005 by Chapman & Hall/CRC

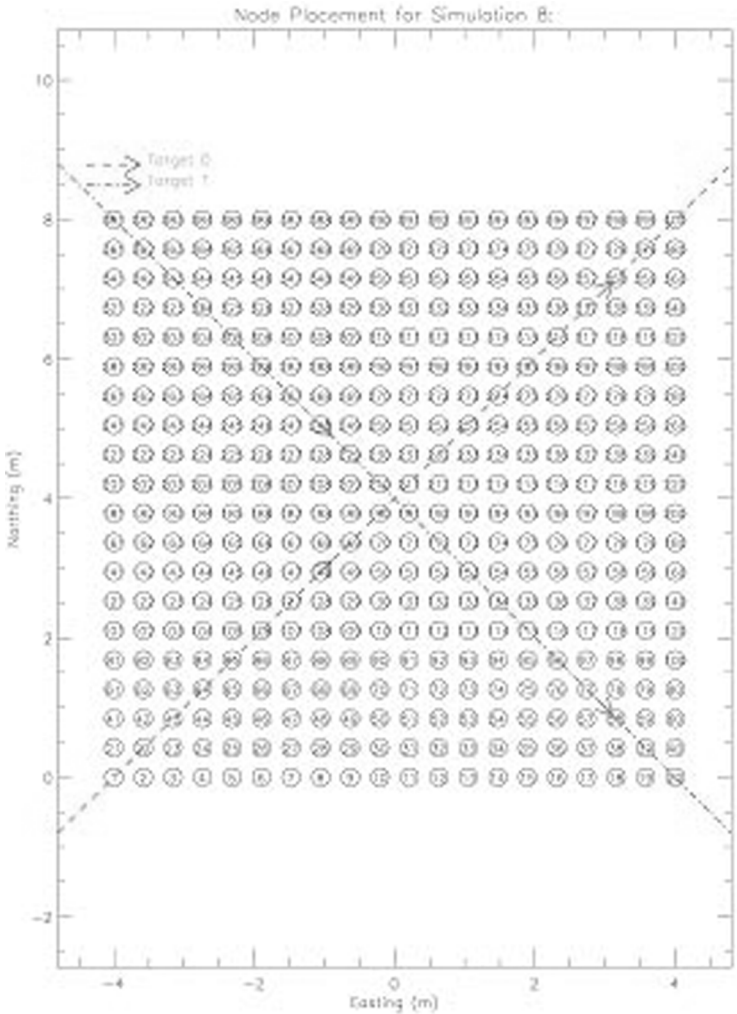


Figure 9.33. Comparison of the two multiple target tracking simulation scenarios. Circles are sensor nodes. The faint lines crossing the node field are the target paths. (a) X path simulation; (b) bowtie path simulation.

The collaborative tracking network performed very well in the X pattern tests due to the linear nature of our track continuations. Tracks seldom jumped to the opposite target path and almost always tracked both targets separately. Bowtie tracking, however, turned out to be more complex. See Figures 9.34 and 9.35.

Bowtie target paths that approach each other too closely at point of nearest approach (Conjunction) tend to cause the tracks to cross-over to the opposite target path as if the targets' paths had crossed each other (Figure 9.34). Again, this is due to the linear nature of the track continuations.

As conjunction distance increases beyond a certain point (critical conjunction), the incidence of cross-over decreases dramatically (Figure 9.35). Minimum effective conjunction is the smallest value of conjunction where the incidence of cross-over begins to decrease to acceptable levels.

According to our analysis as shown in Figure 9.36, if a clump range equal to the node separation is used, then critical conjunction is equal to the node separation and minimum effective conjunction is approximately 1.5 times the node separation. If clump range is equal to two times the node separation,

© 2005 by Chapman & Hall/CRC

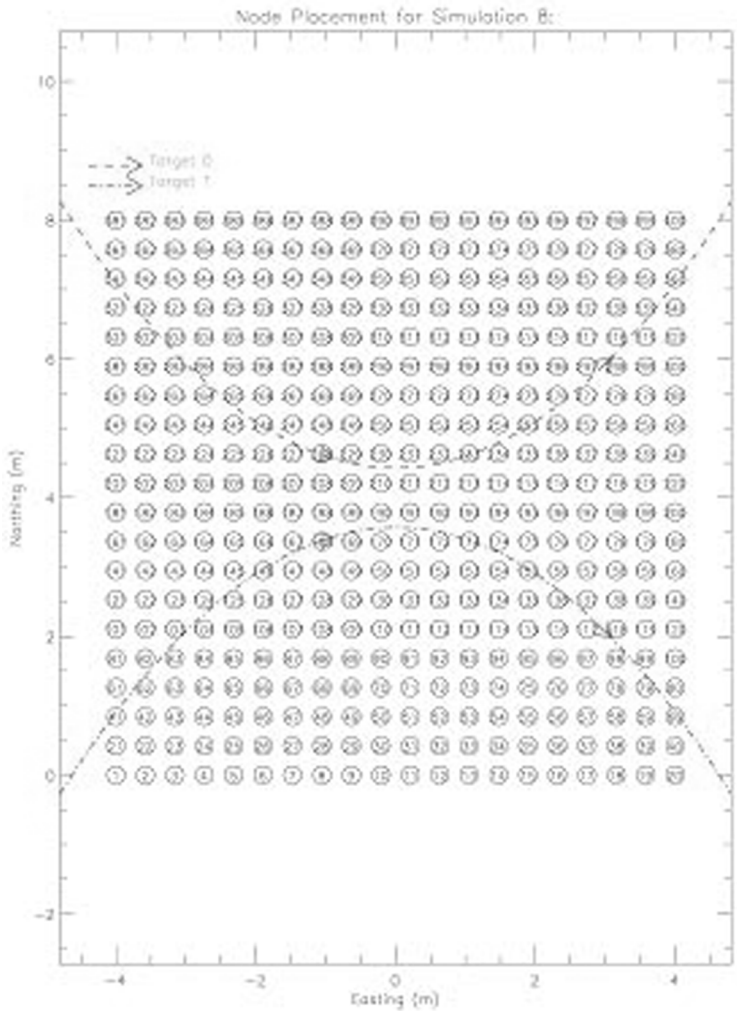


Figure 9.33. Continued.

then critical conjunction is equal to 2.5 times the node separation, and minimum effective conjunction is approximately three times the node separation or 1.5 times the clump range.

The significant result of this analysis seems to be that the minimum effective conjunction is equal to 1.5 times the clump range. This means that ColTraNe should be able to track multiple targets independently provided they are separated by at least 1.5 times the clump range. This appears to be related to fundamental sampling limitations based on Nyquist sampling theory [17].

9.13 Conclusion

This chapter presents a distributed entity-tracking framework that embeds the tracking logic in a self-organized distributed network. Tracking is performed by solving the sub-problems of detection, fusion, association, track formation, and track extrapolation.

© 2005 by Chapman & Hall/CRC

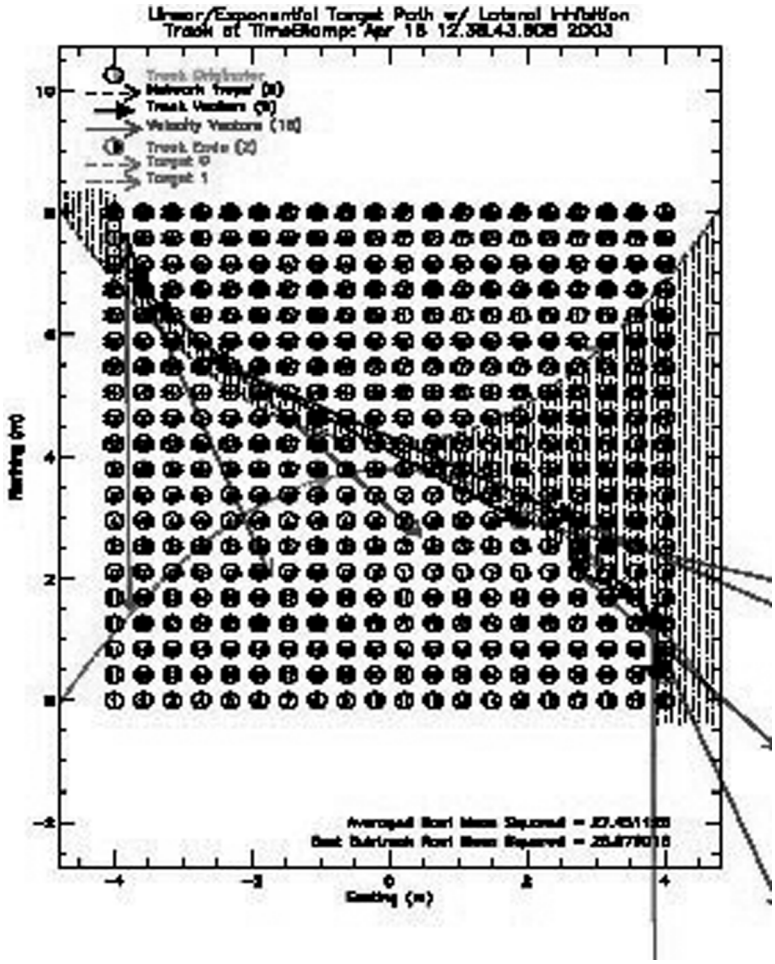


Figure 9.34. Bowtie tracks for conjunction equal to node separation distance. Dark arrows are the reported target tracks. Lighter arrows are the calculated velocity vectors. Shaded areas are the areas between the curves used to determine track error. Other notation as for Figure 9.33. (a) Track for target 1; (b) track for target 2.

Local collaboration determines the values of parameters such as position, velocity, and entity type at points along the track. These variables become state estimates that are used in track formation and data association. Local processing reduces the amount of information that is global and reduces power consumption.

This approach allows us to study tracking as a distributed computation problem. We use in-house CA to construct models based on the interaction of autonomous nodes. These models include system faults and network traffic. We posit that this type of analysis is important for the design of robust distributed systems, like autonomous sensor networks.

Simple CA can be classified into four equivalence classes. Two-dimensional traffic modeling CA are more difficult to classify. The cellular behavior may be periodic, stable, and chaotic in different regions of the CA in question. Exact classification may be impossible or inappropriate.

© 2005 by Chapman & Hall/CRC

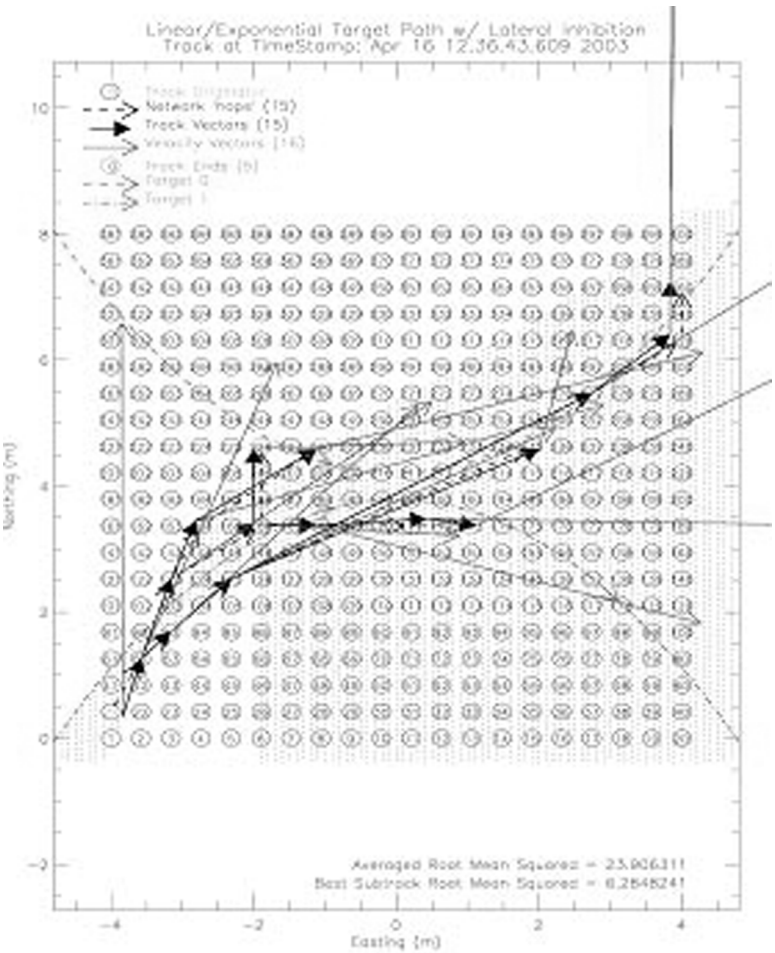


Figure 9.34. Continued.

We have shown that, for certain probabilities of false positives, stable traffic jams will form around the sink location, whereas for other values unstable traffic jams form. These are traffic jams that continue to form, disappear, and reform. This oscillatory behavior is typical of periodic behavior of CA. It is possible to have a stable traffic jam with an unstable boundary.

In the target-tracking context, we have established strong and weak points of the algorithms used. Pheromones appear to be robust, but they transmit more data than the other algorithms. They can also be fooled by false positives.

The Bayesian network is effective for reducing the transmission of false positives, but it has difficulty in maintaining track continuation. Most likely, further work is required to tune the probabilities used.

EKF tracking may not be appropriate for this level of analysis, since it is designed to overcome Gaussian noise. At this level of fidelity that type of noise is less important. The CA model is discrete and the EKF is meant for use with continuous data.

Hybrid approaches may be possible and desirable. One possible avenue to consider is using the Bayesian logic to restrict the propagation of pheromones or to analyze the strength of the pheromone concentration present.

© 2005 by Chapman & Hall/CRC

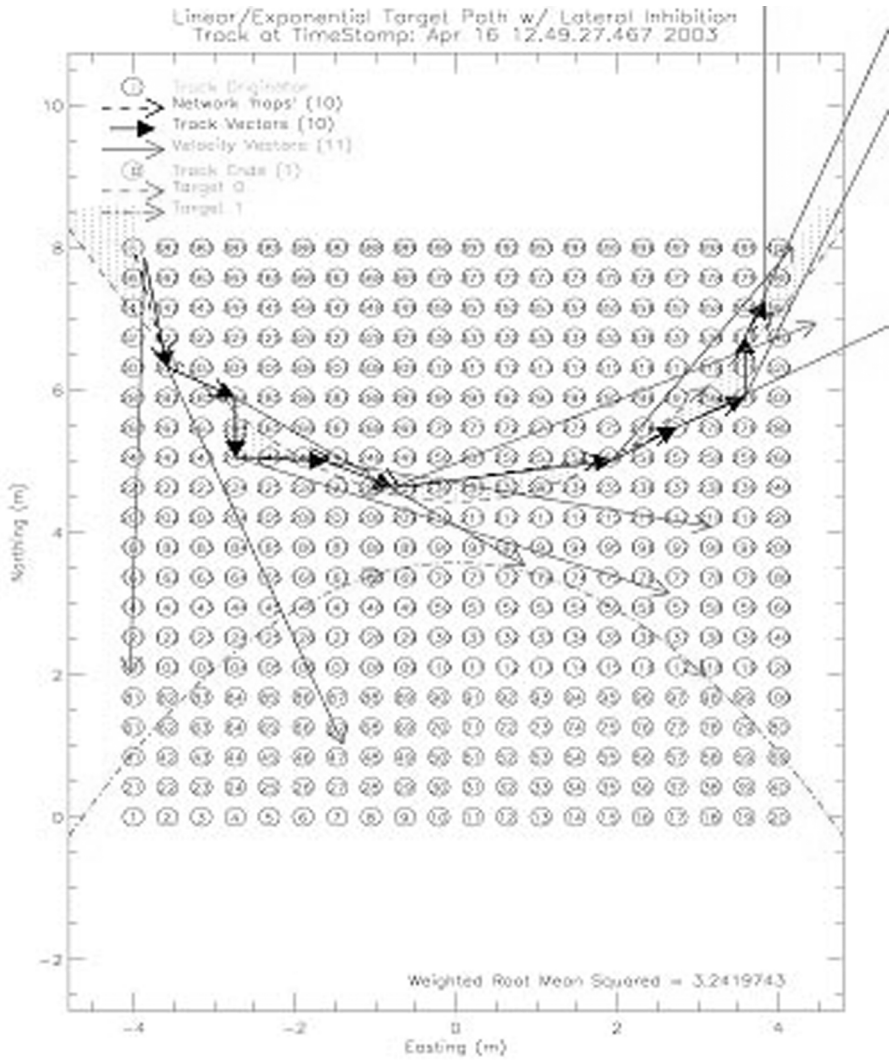


Figure 9.35. Bowtie tracks for conjunction equal to two times node separation distance. Notation as for Figure 9.34. (a) Track for target 1; (b) track for target 2.

Our tracking algorithm development continues by porting these algorithms to a prototype implementation which has been tested in the field. CPA target detections and EKF track continuations are used to track targets through the field with minimal interference from environmental noise. Lateral inhibition is used to enforce some consistency among track association decisions.

Our work indicates to us that performing target tracking in a distributed manner greatly simplifies the multi-target tracking problem. If sensor nodes are dense enough and targets are sparse enough, then the multi-target tracking is a disjoint set of single-target tracking problems. Centralized approaches will also become untenable as target density increases.

A power analysis of our approach versus a centralized approach, such as beamforming, was presented. The power analysis shows that ColTraNe is much more efficient than beamforming for distributed sensing. This is because ColTraNe extracts relevant information from time series locally. It also limits information transmission to the regions that absolutely require the information.

© 2005 by Chapman & Hall/CRC

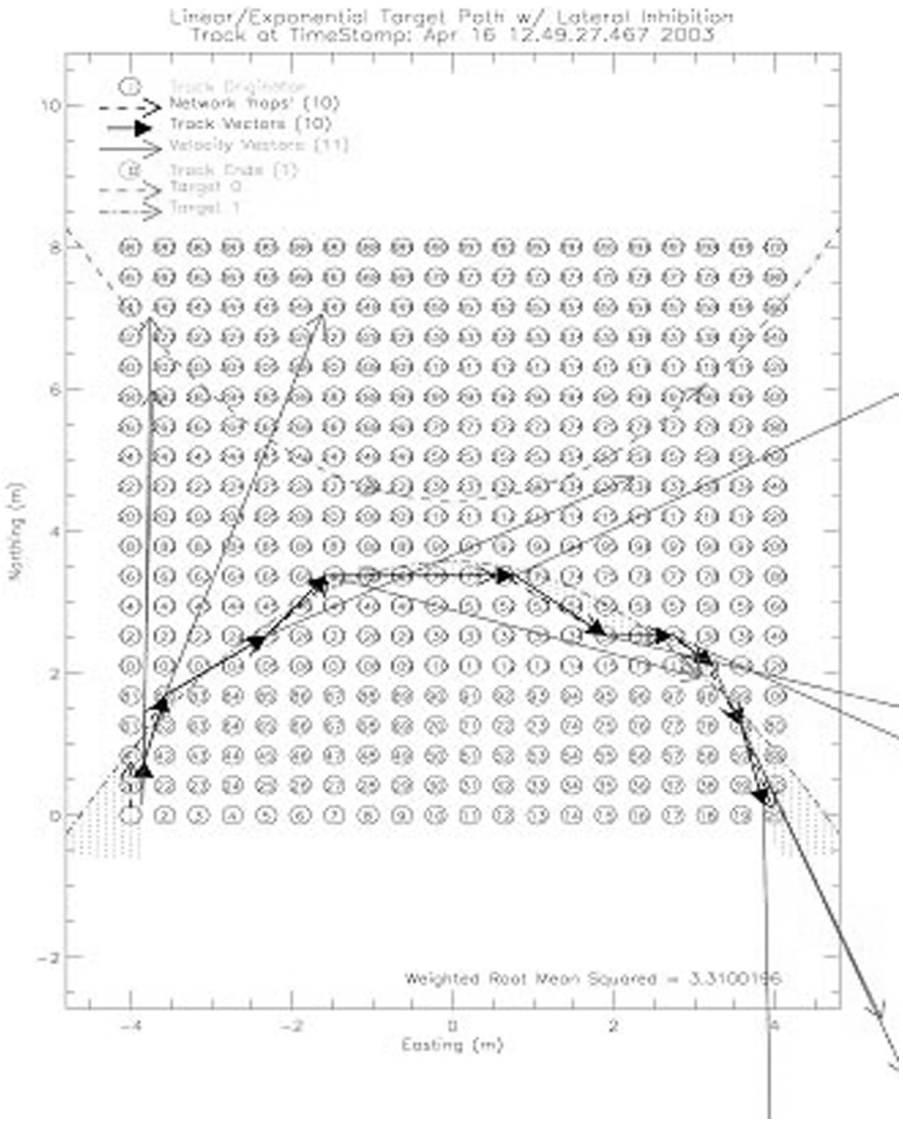


Figure 9.35. Continued.

The chapter concluded with an analysis of the distributed tracker to distinguish multiple targets in a simulated environment. Analysis shows that ColTraNe can effectively track multiple targets provided there is at least two nodes between the target paths at all points, as predicted by Nyquist.

We are continuing our research in distributed sensing applications. Among the topics of interest are:

1. Power-aware methods of assigning system resources.
2. Hybrid tracking methods.
3. Use of symbolic dynamics for inferring target behavior classes.
4. Development of other peer-to-peer distributed behaviors, such as ColTraNe, that are resistant to hardware failures.

© 2005 by Chapman & Hall/CRC

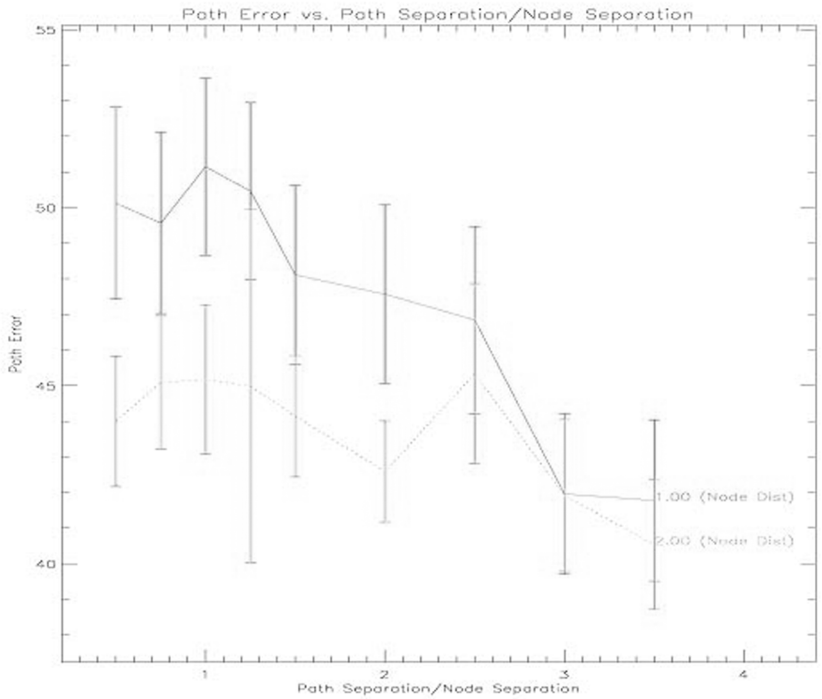


Figure 9.36. Finding critical conjunction experimentally. The darker upper line displays the results when the clump range is equal to the node separation distance. The lighter lower line displays the results when clump range is equal to two times the node separation distance.

Acknowledgments

Efforts were sponsored by the Defense Advance Research Projects Agency (DARPA) Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-99-2-0520 (Reactive Sensor Network) and DARPA and the Space and Naval Warfare Systems Center, San Diego, under grant N66001-00-G8947 (Semantic Information Fusion). The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted to necessarily represent the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), the Air Force Research Laboratory, the U.S. Navy, or the U.S. Government.

References

[1] Intanagonwiwat, C. *et al.*, Directed diffusion: a scalable and robust communication paradigm for sensor networks, in *Mobicom 2000*, Boston, MA, August, 2000, 56.

[2] Brooks, R.R. *et al.*, Reactive sensor networks: mobile code support for autonomous sensor networks, *Distributed Autonomous Robotic Systems DARS 2000*, Springer Verlag, Tokyo, 2000, 471.

[3] USC/ISI, Xerox PARC, LBNL, and UCB, *Virtual Internet Testbed*, http://www.isi.edu/nsnam/vint/.

[4] Blackman S.S. and Broida, T.J., Multiple sensor data association and fusion in aerospace applications, *Journal of Robotic Systems* 7(3), 445, 1990.

[5] Brooks R.R. and Iyengar, S.S., *Multi-Sensor Fusion: Fundamentals and Applications with Software*, Prentice Hall PTR, Upper Saddle River, NJ, 1998.

© 2005 by Chapman & Hall/CRC

[6] Friedlander, D.S. and Phoha, S., Semantic information fusion of coordinated signal processing in mobile sensor networks, *Special Issue on Sensor Networks of the International Journal of High Performance Computing Applications* 16(3), 235, 2002.
[7] Brueckner, S., *Return from the ant: synthetic ecosystems for manufacturing control*, Dr. rer. Nat. Dissertation, Fach Informartik, Humboldt-Universitaet zu Berlin, 2000.
[8] Press, W.H. *et al.*, *Numerical Recipes in C*, 2nd ed., Cambridge University Press, London, UK, 1997.
[9] Pearl, J., Fusion, propagation, and structuring in belief networks, *Artificial Intelligence* 29, 241, 1986.
[10] Wolfram, S., *Cellular Automata and Complexity*, Addison-Wesley, Reading, MA, 1994.
[11] Delorme M. and Mazoyer J. (eds), *Cellular Automata A Parallel Model*, Kluwer Academic PTR, Dordrecht, The Netherlands.
[12] Chowdhury, D. *et al.*, Simulation of vehicular traffic: a statistical physics perspective, *Computing in Science & Engineering* Sept–Oct, 80, 2000.
[13] Portugali, J., *Self-Organization and the City*, Springer Series in Synergetics, Springer Verlag, Berlin, 2000.
[14] Heidemann, J. *et al.*, Building efficient wireless sensor networks with low-level naming, in *Proceedings of Symposium on Operating System Principles* Oct. 2001, 146.
[15] Bar-Shalom Y. and Li, X.-R., *Estimation and Tracking: Principles, Techniques, and Software*, Artech House, Boston, 1993.
[16] Yao K. *et al.*, Blind beamforming on a randomly distributed sensor array system, *IEEE Journal on Selected Areas in Communications* 16, 1555, 1998.
[17] Jacobson, N., Target parameter estimation in a distributed acoustic network, Honors Thesis, The Pennsylvania State University, Spring 2003.
[18] Nagel, K., From particle hopping models to traffic flow theory, in *Traffic Flow Theory Simulation Models, Macroscopic Flow Relationships, and Flow Estimation and Prediction: Transportation Research Record No. 1644*, Transportation Research Board National Research Council, National Academy Press, Washington DC, 1998, 1.

© 2005 by Chapman & Hall/CRC

10

Collaborative Signal and Information Processing: An Information-Directed Approach*

Feng Zhao, Jie Liu, Juan Liu, Leonidas Guibas, and James Reich

10.1 Sensor Network Applications, Constraints, and Challenges

Networked sensing offers unique advantages over traditional centralized approaches. Dense networks of distributed networked sensors can improve perceived signal-to-noise ratio (SNR) by decreasing average distances from sensor to target. Increased energy efficiency in communications is enabled by the multi-hop topology of the network [1]. Moreover, additional relevant information from other sensors can be aggregated during this multi-hop transmission through in-network processing [2]. But perhaps the greatest advantages of networked sensing are in improved robustness and scalability. A decentralized sensing system is inherently more robust against individual sensor node or link failures, because of redundancy in the network. Decentralized algorithms are also far more scalable in practical deployment, and may be the only way to achieve the large scales needed for some applications.

A sensor network is designed to perform a set of high-level information processing tasks, such as detection, tracking, or classification. Measures of performance for these tasks are well defined, including detection, false alarms or misses, classification errors, and track quality. Commercial and military applications include environmental monitoring (e.g. traffic, habitat, security), industrial sensing and diagnostics (e.g. factory, appliances), infrastructure protection (e.g. power grid, water distributions), and battlefield awareness (e.g. multi-target tracking).

Unlike a centralized system, however, a sensor network is subject to a unique set of resource constraints, such as limited on-board battery power and limited network communication bandwidth.

*This work is supported in part by the Defense Advances Research Projects Agency (DARPA) under contract number F30602-00-C-0139 through the Sensor Information Technology Program. The views and conclusions contained herein are those of the authors and should note be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

© 2005 by Chapman & Hall/CRC

In a typical sensor network, each sensor node operates untethered and has a microprocessor and limited amount of memory for signal processing and task scheduling. Each node is also equipped with one or more of acoustic microphone arrays, video or still cameras, IR, seismic, or magnetic sensing devices. Each sensor node communicates wirelessly with a small number of local nodes within the radio communication range.

The current generation of wireless sensor hardware ranges from the shoe-box-sized Sensoria WINS NG sensors [3] with an SH-4 microprocessor to the matchbox-sized Berkeley motes with an eight-bit microcontroller [4]. It is well known that communicating one bit over the wireless medium consumes far more energy than processing the bit. For the Sensoria sensors and Berkeley motes, the ratio of energy consumption for communication and computation is in the range of 1,000–10,000. Despite the advances in silicon fabrication technologies, wireless communication will continue to dominate the energy consumption of embedded networked systems for the foreseeable future [5]. Thus, minimizing the amount and range of communication as much as possible, e.g. through local collaboration, data compression, or invoking only the nodes that are relevant to a given task, can significantly prolong the lifetime of a sensor network and leave nodes free to support multi-user operations.

Traditional signal-processing approaches have focused on optimizing estimation quality for a fixed set of available resources. However, for power-limited and multi-user decentralized systems, it becomes critical to carefully select the embedded sensor nodes that participate in the sensor collaboration, balancing the information contribution of each against its resource consumption or potential utility for other users. This approach is especially important in dense networks, where many measurements may be highly redundant, and communication throughput severely limited. We use the term ''collaborative signal and information processing'' (CSIP) to refer to signal and information processing problems dominated by this issue of selecting embedded sensors to participate in estimation.

This chapter uses tracking as a representative problem to expose the key issues for CSIP — how to determine what needs to be sensed dynamically, who should sense, how often the information must be communicated, and to whom. The rest of the chapter is organized as follows. Section 10.2 will introduce the tracking problem and present a set of design considerations for CSIP applications. Sections 10.3 and 10.4 will analyze a range of tracking problems that differ in the nature of the information being extracted, and describe and compare several recent contributions that adopted information-based approaches. Section 10.5 will discuss future directions for CSIP research.

10.2 Tracking as a Canonical Problem for CSIP

Tracking is an essential capability in many sensor network applications, and is an excellent vehicle to study information organization problems in CSIP. It is especially useful for illustrating a central problem of CSIP: dynamically defining and forming sensor groups based on task requirements and resource availability.

From a sensing and information processing point of view, we define a sensor network as a tuple, $\mathrm{Sn} = \langle V, E, P_V, P_E \rangle$. V and E specify a network graph, with its nodes V, and link connectivity $E \subseteq V \times V$. P_V is a set of functions which characterizes the properties of each node in V, including its location, computational capability, sensing modality, sensor output type, energy reserve, and so on. Possible sensing modalities includes acoustic, seismic, magnetic, IR, temperature, or light. Possible output types include information about signal amplitude, source direction-of-arrival (DOA), target range, or target classification label. Similarly, P_E specifies properties for each link such as link capacity and quality.

A tracking task can be formulated as a constrained optimization problem $\mathrm{Tr} = \langle \mathrm{Sn}, \mathrm{Tg}, \mathrm{Sm}, Q, O, C \rangle$. Sn is the sensor network specified above. Tg is a set of targets, specifying for each target the location, shape (if not a point source), and signal source type. Sm is a signal model for how the target signals propagate and attenuate in the physical medium. For example, a possible power attenuation model for an acoustic signal is the inverse distance squared model. Q is a set of user queries, specifying query instances and query entry points into the network. A sample query is ''Count the number of targets in

© 2005 by Chapman & Hall/CRC

region *R*." *O* is an objective function, defined by task requirements. For example, for a target localization task, the objective function could be the localization accuracy, expressed as the trace of the covariance matrix for the position estimate. $C = \{C_1, C_2, \dots\}$ specifies a set of constraints. An example is localizing an object within a certain amount of time and using no more than a certain quantity of energy. The constrained optimization finds a set of feasible sensing and communication solutions for the problem that satisfies the given set of constraints. For example, a solution to the localization problem above could be a set of sensor nodes on a path that gathers and combines data and routes the result back to the querying node.

In wireless sensor networks, some of the information defining the objective function and/or constraints is only available at run time. Furthermore, the optimization problem may have to be solved in a decentralized way. In addition, anytime algorithms are desirable, because constraints and resource availability may change dynamically.

10.2.1 A Tracking Scenario

We use the following tracking scenario (Figure 10.1) to bring out key CSIP issues. As a target *X* moves from left to right, a number of activities occur in the network:

1. *Discovery*. Node *a* detects *X* and initiates tracking.
2. *Query processing*. A user query *Q* enters the network and is routed towards regions of interest, in this case the region around node *a*. It should be noted that other types of query, such as long-running query that dwell in a network over a period of time, are also possible.
3. *Collaborative processing*. Node *a* estimates the target location, possibly with help from neighboring nodes.

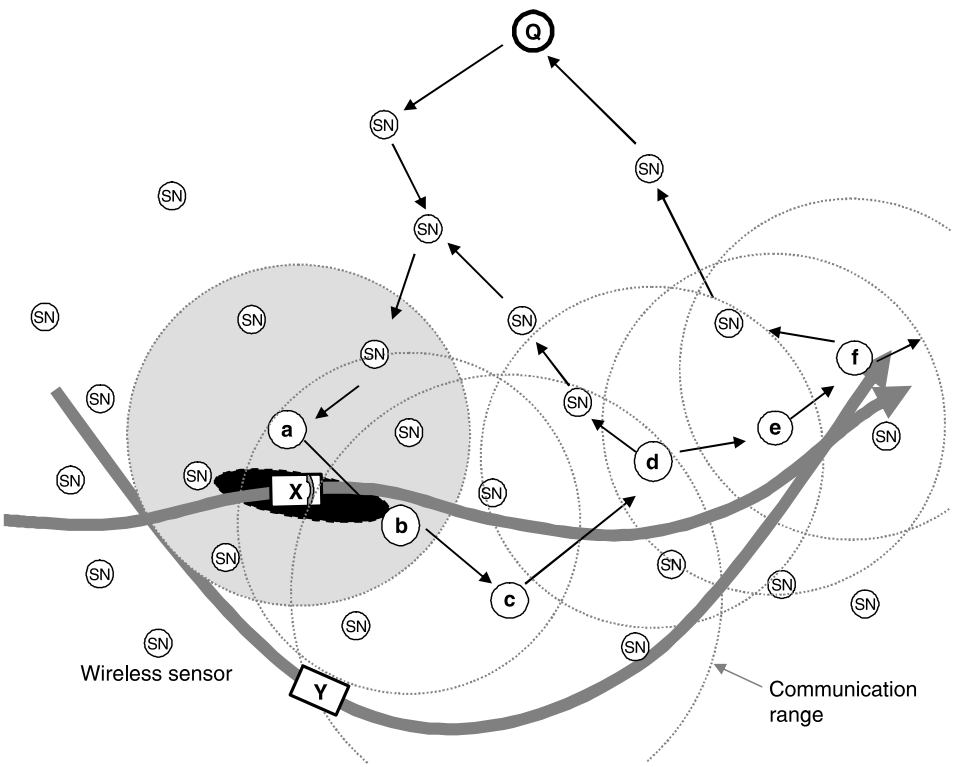


Figure 10.1. A tracking scenario, showing two moving targets, *X* and *Y*, in a field of sensors. Large circles represent the range of radio communication from each node.

© 2005 by Chapman & Hall/CRC

4. *Communication*. Node *a* may hand off data to node *b*, *b* to *c*, etc.
5. *Reporting*. Node *d* or *f* summarizes track data and sends it back to the querying node.

Let us now assume another target, *Y*, enters the region around the same time. The network will have to handle multiple tasks in order to track both targets simultaneously. When the two targets move close to each other, the problem of properly associating a measurement to a target track, the so-called *data association problem*, becomes tricky. In addition, collaborative sensor groups, as defined earlier, must be selected carefully, since multiple groups might need to share the same physical hardware [6].

This tracking scenario raises a number of fundamental information-processing problems in distributed information discovery, representation, communication, storage, and querying: (1) in collaborative processing, the issues of target detection, localization, tracking, and sensor tasking and control; (2) in networking, the issues of data naming, aggregation, and routing; (3) in databases, the issues of data abstraction and query optimization; (4) in human–computer interface, the issues of data browsing, search, and visualization; (5) in software services, the issues of network initialization and discovery, time and location services, fault management, and security. In the rest of the chapter, we will focus on the collaborative processing aspects and touch on other issues only as necessary.

A common task for a sensor network is to gather information from the environment. Doing this under the resource constraints of a sensor network may require data-centric routing and aggregation techniques which differ considerably from TCP/IP end-to-end communication. Consequently, the research community has been searching for the right "sensor net stack" that can provide suitable abstractions over networking and hardware resources. While defining a unifying architecture for sensor networks is still an open problem, we believe a key element of such an architecture is the *principled interaction between the application and networking layers*. For example, Section 10.3 will describe an approach that expresses application requirements as a set of information and cost constraints so that an *ad hoc* networking layer using, for example, the diffusion routing protocol [2], can effectively support the application.

10.2.2 Design Desiderata in Distributed Tracking

In essence, a tracking system attempts to recover the state of a target (or targets) from observations. Informally, we refer to the information about the target state distilled from measurement data as a belief or belief state. An example is the posterior probability distribution of target state, as discussed in Section 10.3. As more observation data are available, the belief may be refined and updated.

In sensor networks, the belief state can be stored centrally at a fixed node, at a sequence of nodes through successive hand-offs, or at a set of nodes concurrently. In the first case (Figure 10.2(a)), a fixed node is designated to receive measurements from other relevant sensors through communication. This simpler tracker design is obtained at the cost of potentially excessive communication and reduced robustness to node failure. It is feasible only for tracking nearly stationary targets, and is in general neither efficient nor scalable.

In the second case (Figure 10.2(b)), the belief is stored at a node called the leader node, which collects data from nearby, relevant sensors. As the phenomenon of interest moves or environmental conditions vary, the leadership may change hands among sensor nodes. Since the changes in physical conditions are often continuous in nature, these handoffs often occur within a local geographic neighborhood. This moving leader design localizes communication, reducing overall communication and increasing the lifetime of the network. The robustness of this method may suffer from potential leader node attrition, but this can be mitigated by maintaining copies of the belief in nearby nodes and detecting and responding to leader failure. The key research challenge for this design is to define an effective selection criterion for sensor leaders, to be addressed in Section 10.3.

Finally, the belief state can be completely distributed across multiple sensor nodes (Figure 10.2(c)). The inference from observation data is accomplished nodewise, thus localizing the communication. This is attractive from the robustness point of view. The major design challenge is to infer global

© 2005 by Chapman & Hall/CRC

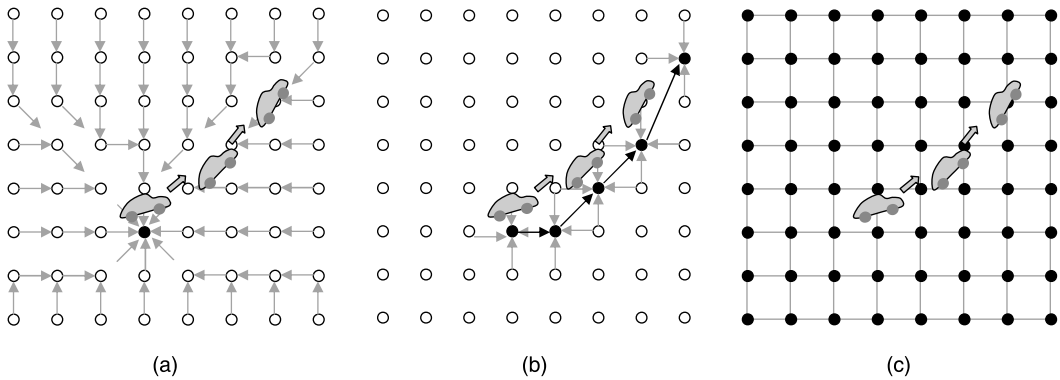


Figure 10.2. Storage and communication of target state information in a networked distributed tracker. Circles on the grid represent sensor nodes, and some of the nodes, denoted by solid circles, store target state information. Thin, faded arrows or lines denote communication paths among the neighbor nodes. Thin, dark arrows denote sensor hand-offs. A target moves through the sensor field, indicated by thick arrows. (a) A fixed single leader node has the target state. (b) A succession of leader nodes is selected according to information such as vehicle movement. (c) Every node in the network stores and updates target state information.

properties about targets efficiently, some of which may be discrete and abstract, from partial, local information, and to maintain information consistency across multiple nodes. Section 10.4 addresses the challenge. Many issues about leaderless distributed trackers are still open and deserve much attention from the research community.

10.3 Information-Driven Sensor Query: A CSIP Approach to Target Tracking

Distributed tracking is a very active field, and it is beyond the scope of this chapter to provide a comprehensive survey. Instead, we will focus on the information processing aspect of the tracking problems, answering questions such as what information is collected by the sensors, how that information is aggregated in the network, and what high-level user queries are answered. This section describes an information-driven sensor query (IDSQ), a set of information-based approaches to tracking individual targets, and discusses major issues in designing CSIP solutions. Next, Secction 10.4 presents approaches to other tracking problems, where the focus is more on uncovering abstract and discrete target properties, such as target density, rather than just their locations.

10.3.1 Tracking Individual Targets

The basic task of tracking a moving target in a sensor field is to determine and report the underlying target state $\mathbf{x}^{(t)}$, such as its position and velocity, based on the sensor measurements up to time t, denoted as $\overline{\mathbf{z}^{(t)}} = \{\mathbf{z}^{(0)}, \mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(t)}\}$. Many approaches have been developed over the last half century. These include Kalman filters, which assume a Gaussian observation model and linear state dynamics, and, more generally, sequential Bayesian filtering, which computes the posterior belief at time $t+1$ based on the new measurement $\mathbf{z}^{(t+1)}$ and the belief $p(\mathbf{x}^{(t)}|\overline{\mathbf{z}^{(t)}})$ inherited from time t:

$$p(\mathbf{x}^{(t+1)}|\overline{\mathbf{z}^{(t+1)}}) \propto p(\mathbf{z}^{(t+1)}|\mathbf{x}^{(t+1)}) \cdot \int p(\mathbf{x}^{(t+1)}|\mathbf{x}^{(t)}) \cdot p(\mathbf{x}^{(t)}|\overline{\mathbf{z}^{(t)}})\, d\mathbf{x}^{(t)}$$

Here, $p(\mathbf{z}^{(t+1)}|\mathbf{x}^{(t+1)})$ denotes the observation model and $p(\mathbf{x}^{(t+1)}|\mathbf{x}^{(t)})$ the state dynamics model. As more data are gathered over time, the belief $p(\mathbf{x}^{(t)}|\overline{\mathbf{z}^{(t)}})$ is successively refined.

© 2005 by Chapman & Hall/CRC

Kalman filters and many practical forms of Bayesian filter assume that the measurement noise across multiple sensors is independent, which is not always the case. Algorithms, such as covariance intersection, have been proposed to combine data from sensors with correlated information. Although these methods have been successfully implemented in applications, they were primarily designed for centralized platforms. Relatively little consideration was given to the fundamental problems of moving data across sensor nodes in order to combine data and update track information. There was no cost model for communication in the tracker. Furthermore, owing to communication delays, sensor data may arrive at a tracking node out of order compared with the original time sequence of the measurements. Kalman or Bayesian filters assume a strict temporal order on the data during the sequential update, and may have to roll back the tracker in order to incorporate "past" measurements, or throw away the data entirely.

For multi-target tracking, methods such as multiple hypothesis tracking (MHT) [7] and joint probabilistic data association (JPDA) [8] have been proposed. They addressed the key problem of data association, of pairing sensor data with targets, thus creating association hypotheses. MHT forms and maintains multiple association hypotheses. For each hypothesis, it computes the probability that it is correct. On the other hand, JPDA evaluates the association probabilities and combines them to compute the state estimate. Straightforward applications of MHT and JPDA suffer from a combinatorial explosion in data association. Knowledge about targets, environment, and sensors can be exploited to rank and prune hypotheses [9,10].

10.3.2 Information-Based Approaches

The main idea of information-based approaches is to base sensor collaboration decisions on information content, as well as constraints on resource consumption, latency, and other costs. Using information utility measures, sensors in a network can exploit the information content of data already received to optimize the utility of future sensing actions, thereby efficiently managing scarce communication and processing resources. The distributed information filter, as described by Manyika and Durrant-Whyte [11], is a global method requiring each sensor node to communicate its measurement to a central node where estimation and tracking are carried out. In this method, sensing is distributed and tracking is centralized. Directed-diffusion routes sensor data in a network to minimize communication distance between data sources and data sinks [2, 12]. This is an interesting way of organizing a network to allow publish-and-subscribe to occur at a very fine grained level. A prediction-based tracking algorithm is described by Brooks *et al.* [13] which uses estimates of target velocity to select which sensors to query. An IDSQ [14,15] formulates the tracking problem as a more general distributed constrained optimization that maximizes information gain of sensors while minimizing communication and resource usage. We describe the main elements of an IDSQ here.

Given the current belief state, we wish to update the belief incrementally by incorporating the measurements of other nearby sensors. However, not all available sensors in the network provide useful information that improves the estimate. Furthermore, some information may be redundant. The task is to select an optimal subset and an optimal order of incorporating these measurements into our belief update. Note that, in order to avoid prohibitive communication costs, this selection must be done without explicit knowledge of measurements residing at other sensors. The decision must be made solely based upon known characteristics of other sensors, such as their position and sensing modality, and predictions of their contributions, given the current belief.

Figure 10.3 illustrates the basic idea of optimal sensor selection. The illustration is based upon the assumption that estimation uncertainty can be effectively approximated by a Gaussian distribution, illustrated by uncertainty ellipsoids in the state space. In the figure, the solid ellipsoid indicates the belief state at time t, and the dashed ellipsoids are the incrementally updated belief after incorporating an additional measurement from a sensor, S1 or S2, at the next time step. Although in both cases, S1 and S2, the area of high uncertainty is reduced by 50%, the residual uncertainty of the S2 case is not reduced

© 2005 by Chapman & Hall/CRC

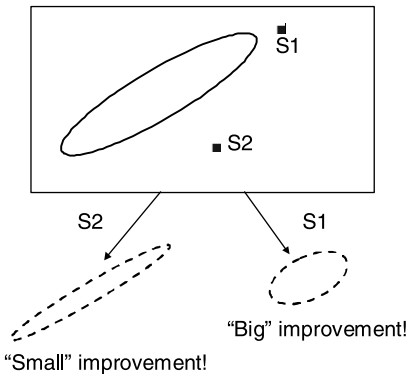


Figure 10.3. Sensor selection based on information gain of individual sensor contributions. The information gain is measured by the reduction in the error ellipsoid. In the figure, reduction along the longest axis of the error ellipsoid produces a larger improvement in reducing uncertainty. Sensor placement geometry and sensing modality can be used to compare the possible information gain from each possible sensor selection, S1 or S2.

along the long principal axis of the ellipse. If we were to decide between the two sensors, then we might favor case S1 over case S2, based upon the underlying measurement task.

In distributed sensor network systems we must balance the information contribution of individual sensors against the cost of communicating with them. For example, consider the task of selecting among K sensors with measurements $\{\mathbf{z}_i\}_{i=1}^{K}$. Given the current belief $p(\mathbf{x} \mid \{z_i\}_{i\in U})$, where $U \subset \{1, \ldots, K\}$ is the subset of sensors whose measurement has already been incorporated, the task is to choose which sensor to query among the remaining unincorporated set $A = \{1, \ldots, K\} \setminus U$. For this task, an objective function as a mixture of information and cost is designed in [15]:

$$O\Big(p\Big(\mathbf{x}|\overline{\mathbf{z}_j^{(t)}}\Big)\Big) = \alpha\phi\Big(p\Big(\mathbf{x}|\overline{\mathbf{z}_{j-1}^{(t)}}, \mathbf{z}_j^{(t)}\Big)\Big) - (1-\alpha)\psi(\mathbf{z}_j^{(t)}) \tag{10.1}$$

Here, ϕ measures the information utility of incorporating the measurement $\mathbf{z}_j^{(t)}$ from sensor j, ψ is the cost of communication and other resources, and α is the relative weighting of the utility and cost. With this objective function, the sensor selection criterion takes the form

$$\hat{j} = \arg\max_{j\in A} O(p(\mathbf{x}|\{\mathbf{z}_i\}_{i\in U} \cup \{z_j\})) \tag{10.2}$$

This strategy selects the best sensor given the current state $p(x \mid \{z_i\}_{i\in U})$. A less greedy algorithm has been proposed by Liu *et al.* [16], extending the sensor selection over a finite look-ahead horizon.

Metrics of information utility ϕ and cost ψ may take various forms, depending on the application and assumptions [14]. For example, Chu *et al.* [15] considered the query routing problem: assuming a query has entered from a fixed node, denoted by "?" in Figure 10.4, the task is to route the query to the target vicinity, collect information along an optimal path, and report back to the querying node. Assuming the belief state is well approximated by a Gaussian distribution, the usefulness of the sensor data (in this case, range data) ϕ is measured by how close the sensor is to the mean of the belief state under a Mahalanobis metric, assuming that close-by sensors provide more discriminating information. The cost ψ is given here by the squared Euclidean distance from the sensor to the current leader, a simplified model of the energy expense of radio transmission for some environments. The optimal path results from the tradeoff between these two terms. Figure 10.4 plots such a sample path. Note that the

© 2005 by Chapman & Hall/CRC

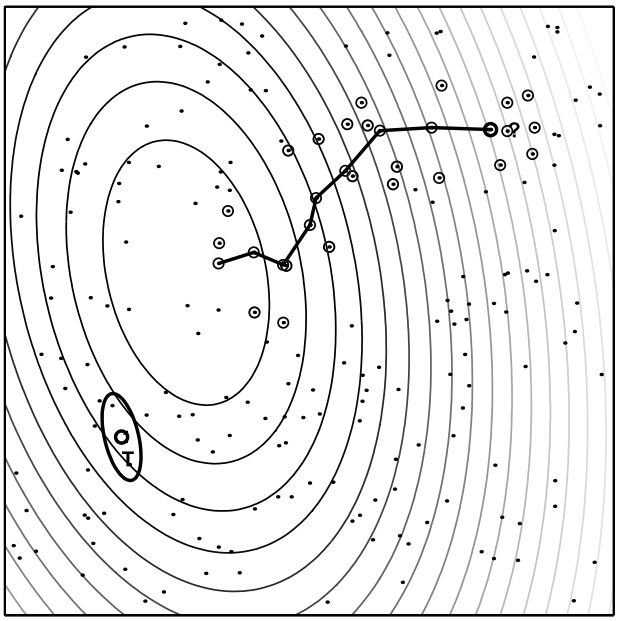


Figure 10.4. Sensor querying and data routing by optimizing an objective function of information gain and communication cost, whose iso-contours are shown as the set of concentric ellipses. The circled dots are the sensors being queried for data along the querying path. "T" represents the target position and "?" denotes the position of the query origin.

belief is updated incrementally along the information collection path. The ellipses in Figure 10.4 show a snapshot of the objective function that an active leader node evaluates locally at a given time step.

For multi-modal non-Gaussian distributions, a mutual information-based sensor selection criterion has been developed and successfully tested on real data [17]. The problem is as follows: assuming that a leader node holds the current belief $p(x^{(t)}|\overline{\mathbf{z}^{(t)}})$, and the cost to query any sensor in its neighborhood $\mathcal{N}$ is identical (e.g. over a wired network or using a fixed power-level radio), the leader selects from $\mathcal{N}$ the most informative sensor to track the moving target. In this scenario, the selection criterion of Equation (10.2) takes the form

$$\hat{j}_{\text{IDSQ}} = \arg\max_{j \in \mathcal{N}} I(X^{(t+1)}; Z_j^{(t+1)} | \overline{Z^{(t)}} = \overline{\mathbf{z}^{(t)}}) \tag{10.3}$$

where $I(\cdot\,;\cdot)$ measures the mutual information in bits between two random variables. Essentially, this criterion selects a sensor whose measurement $\mathbf{z}_j^{(t+1)}$, combined with the current measurement history $\overline{\mathbf{z}^{(t)}}$, would provide the greatest amount of information about the target location $\mathbf{x}^{(t+1)}$. The mutual information can be interpreted as Kullback–Leibler divergence between the beliefs after and before applying the new measurement $\mathbf{z}_j^{(t+1)}$. Therefore, this criterion favors the sensor which, on average, gives the greatest change to the current belief.

To analyze the performance of the IDSQ tracker, we measure how the tracking error varies with sensor density through simulation. Figure 10.5 shows that, as the sensor density increases, the tracking error, expressed as the mean error of the location estimate, decreases, as one would expect, and tends to a floor dominated by sensor noise. This indicates that there is a maximum density beyond which using more sensors gains very little in tracking accuracy.

The IDSQ tracker has been successfully tested in a DARPA tracking experiment at 29 Palms, November 2001. In the experiment, 21 Sensoria WINS NG wireless sensors were used to collect acoustic data from moving vehicles. Details of the results can be found in [17].

© 2005 by Chapman & Hall/CRC

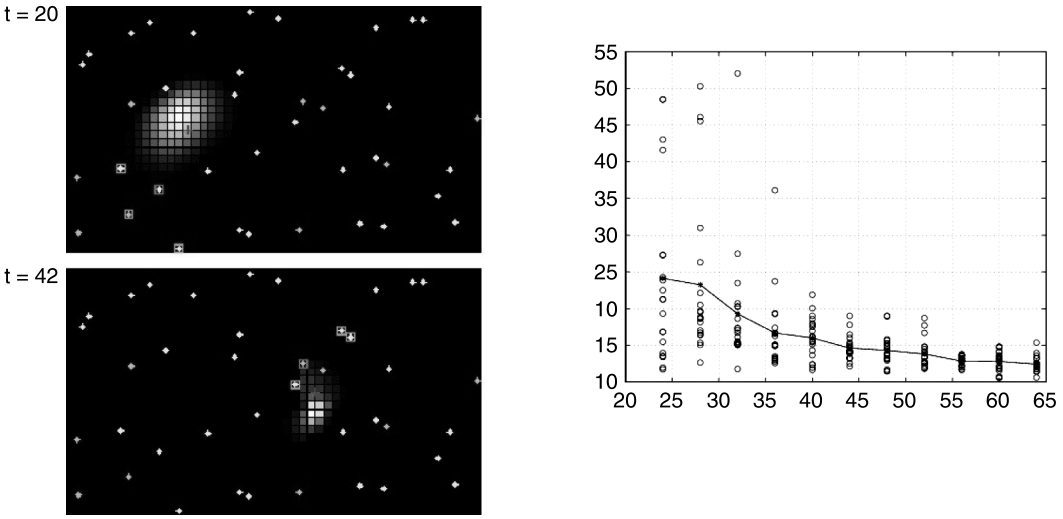


Figure 10.5. Experimental results (right figure) show how the tracking error (vertical axis), defined as the mean error of estimated target positions, varies with the sensor density (horizontal axis), defined as the number of sensors in the sensor field. The left figure shows snapshots of a belief "cloud" — the probability density function of the location estimate — for different local sensor densities.

10.4 Combinatorial Tracking Problems

The discussion of tracking so far has focused on localizing targets over time. In many applications, however, the phenomenon of interest may not be the exact locations of individual objects, but global properties regarding a collection of objects, e.g. the number of targets, their regions of influence, or their boundaries. The information to be extracted in this case may be more discrete and abstract, and may be used to answer high-level queries about the world-state or to make strategic decisions about actions to take.

An expensive way to compute such global class properties of objects is to locate and identify each object in the collection, determine its individual properties, and combine the individual information to form the global answer, such as the total number of objects in the collection. However, in many cases, these class properties can be inferred without accurate localization or identification of all the objects in question. For example, it may be possible to focus on attributes or relations that can be directly sensed by the sensors. This may both make the tracking results more robust to noise and may simplify the algorithms to the point where they can be implemented on less powerful sensor nodes. We call these approaches *combinatorial tracking*.

10.4.1 Counting the Number of Targets

Target counting is an attempt to keep track of the number of distinct targets in a sensor field, even as they move, cross-over, merge, or split. It is representative of a class of applications that need to monitor intensity of activities in an area. To describe the problem, let us consider counting multiple targets in a two-dimensional sensor field, as shown in Figure 10.6. We assume that targets are point-source acoustic signals and can be stationary or moving at any time, independent of the state of other targets. Sensors measure acoustic power and are time synchronized to a global clock. We assume that signals from two targets simply add at a receiving sensor, which is reasonable for noncoherent interference between acoustic sources.

The task here is to determine the number of targets in the region. One way to solve the problem is to compute an initial count and then update the count as targets move, enter, or leave the region. Here, we

© 2005 by Chapman & Hall/CRC

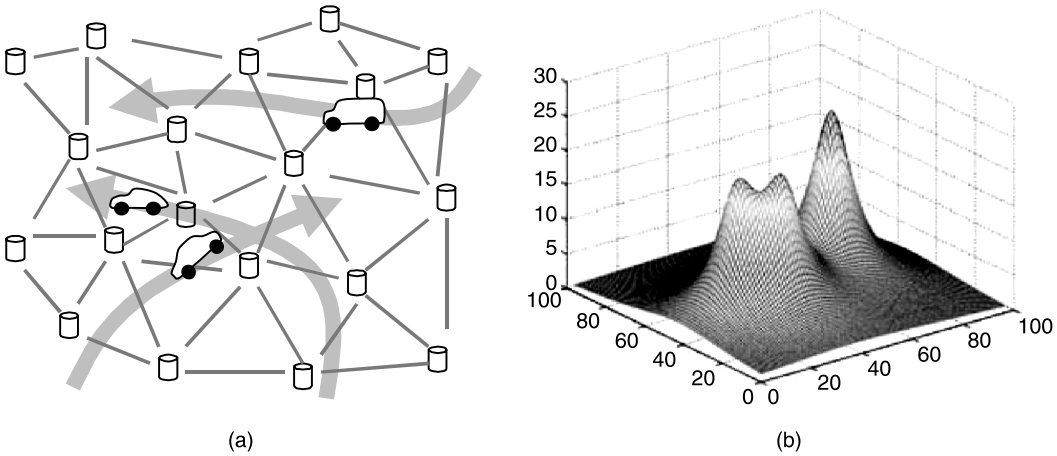


Figure 10.6. Target counting scenario, showing three targets in a sensor field (a). The goal is to count and report the number of distinct targets. With the signal field plotted in (b), the target counting becomes a peak counting problem.

describe a leader-based counting approach, where a sensor leader is elected for each distinct target. A leader is initialized when a target moves into the field. As the target moves, the leadership may switch between sensor nodes to reflect the state change. When a target moves out of the region, the corresponding leader node is deactivated. Note here that the leader election does not rely on accurate target localization, as will be discussed later. The target count is obtained by noting the number of active leader nodes in the network (and the number of targets each is responsible for). Here, we will focus on the leader election process, omitting details of signal and query processing.

Since the sensors in the network only sense signal energy, we need to examine the spatial characteristics of target signals when multiple targets are in close proximity to each other. In Figure 10.6(b), the three-dimensional surface shown represents total target signal energy. Three targets are plotted, with two targets near each other and one target well separated from the rest of the group.

There are several interesting observations to make here:

1. Call the set of sensors that can ''hear'' a target the target influence area. When targets' influence areas are well separated, target counting can be considered as a clustering and a cluster leader election problem. Otherwise, it becomes a peak counting problem.
2. The target signal propagation model has a large impact on target ''resolution.'' The faster the signal attenuates with distance from the source, the easier it is to discern targets from neighboring targets based on the energy of signals they emit.
3. Sensor spacing is also critical in obtaining correct target count. Sensor density has to be sufficient to capture the peaks and valleys of the underlying energy field, yet very densely packed sensors are often redundant, wasting resources.

A decentralized algorithm was introduced for the target counting task [10]. This algorithm forms equivalence classes among sensors, elects a leader node for each class based on the relative power detected at each sensor, and counts the number of such leaders. The algorithm comprises a decision predicate P which, for each node i, tests if it should participate in an equivalence class and a message exchange schema M about how the predicate P is applied to nodes. A node determines whether it belongs to an equivalence class based on the result of applying the predicate to the data of the node, as well as on information from other nearby nodes. Equivalence classes are formed when the process converges. This protocol finds equivalence classes even when multiple targets interfere.

© 2005 by Chapman & Hall/CRC

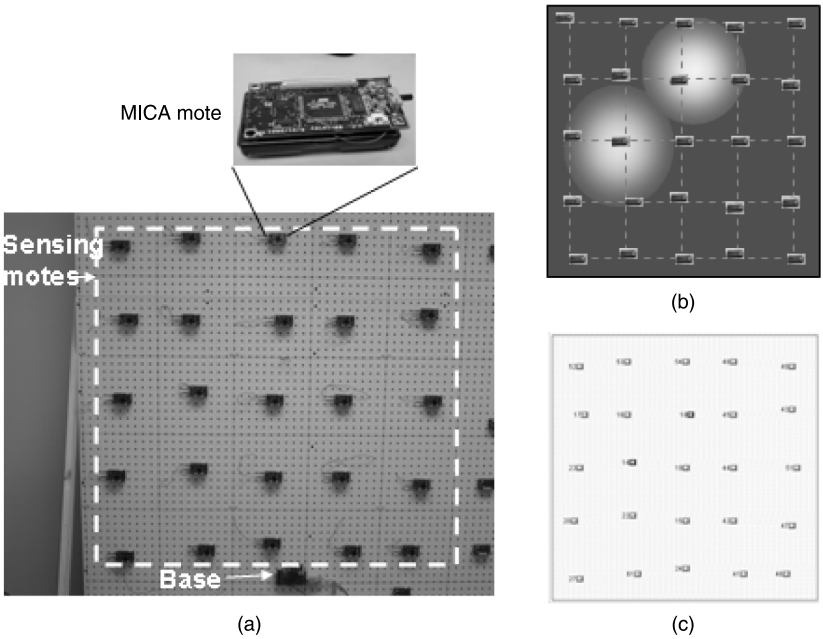


Figure 10.7. Target counting application implemented on Berkeley motes: (a) 25 MICA motes with light sensors are placed on a perturbed grid in a dark room; (b) two light blobs emulating $1/r^2$ signal attenuation are projected onto the mote board; (c) the leader of each collaboration group sends its location back to a base station GUI.

This leader election protocol is very powerful, yet it is lightweight enough to be implemented on sensor nodes such as the Berkeley motes. Figure 10.7 shows an experiment consists of 25 MICA motes with light sensors. The entire application, including code for collaborative leader election and multi-hop communication to send the leader information back to the base station, takes about 10K bytes memory space on a mote.

10.4.2 Contour Tracking

Contour tracking is another example of finding the influence regions of targets without locating them. For a given signal strength, the tracking results are a set of contours, each of which contains one or more targets.

As in the target counting scenario, let us consider a two-dimensions sensor field and point-source targets. One way of determining the contours is by building a mesh over distributed sensor nodes via a Delaunay triangulation or a similar algorithm. The triangulation can be computed offline when setting up the network. Nodes that are connected by an edge of a triangle are called direct neighbors. Given a measurement threshold σ, which defines a σ-contour, a node is called a contour node if it has a sensor reading above σ and at least one of its direct neighbors has a sensor reading below σ. For a sufficiently smooth contour and dense sensor network, a contour can be assumed to intersect an edge only once, and a triangle at exactly two edges, as shown in Figure 10.8. By following this observation, we can traverse the contour by "walking" along the contour nodes. Again, purely local algorithms exist to maintain these contours as the targets move.

10.4.3 Shadow Edge Tracking

Contour tracking can be viewed as a way to determine the boundary of a group of targets. In an extreme case, the group of targets can be a continuum over space, where no single sensor alone can determine

© 2005 by Chapman & Hall/CRC

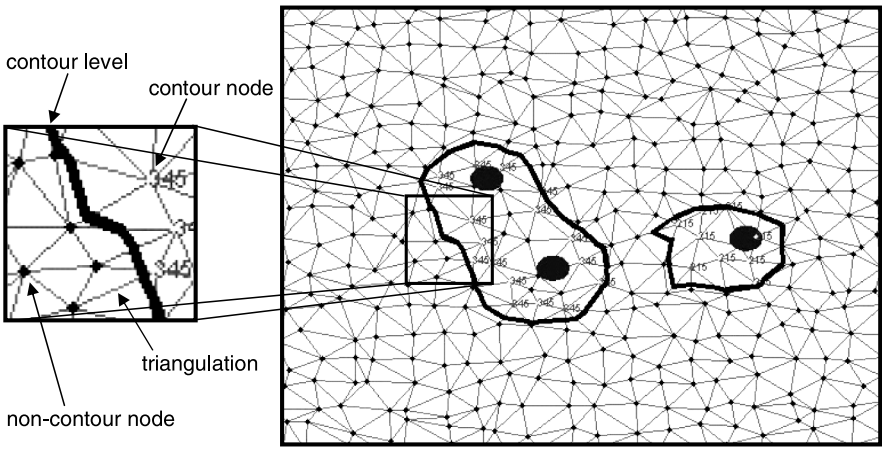


Figure 10.8. Simulation result showing contours for three point targets in a sensor field. The contours are constructed using a distributed marching-squares-like algorithm and are updated as targets move.

the global information from its local measurement. An example of this is to determine and track the boundary of a large object moving in a sensor field, where each sensor only "sees" a portion of the object. One such application is tracking a moving chemical plume over an extended area using airborne and ground chemical sensors.

We assume the boundary of the object is a polygon made of line segments. Our approach is to convert the problem of estimating and tracking a nonlocal (possibly very long) line segment into a local problem using a dual-space transformation [19]. Just as a Fourier transform maps a global property of a signal, such as periodicity in the time domain, to a local feature in the frequency domain, the dual-space transform maps a line in the primal space into a point in the dual space, and *vice versa* (Figure 10.9). Using a primal–dual transformation, each edge of a polygonal object can be tracked as a point in the

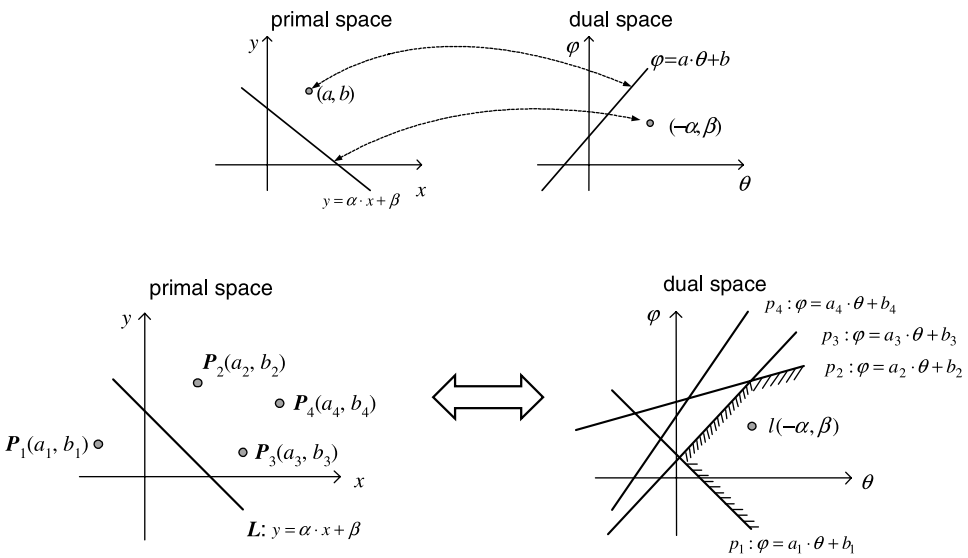


Figure 10.9. Primal–dual transformation, a one-to-one mapping where a point maps to a line and a line maps to a point (upper figure). The image of a half-place shadow edge in the dual space is a point located in a cell formed by the duals of the sensor nodes (lower figure).

© 2005 by Chapman & Hall/CRC

dual space. A tracking algorithm has been developed based on the dual-space analysis and implemented on the Berkeley motes [19]. A key feature of this algorithm is that it allows us to put to sleep all sensor nodes except those in the vicinity of the object boundary, yielding significant energy savings.

Tracking relations among a set of objects is another form of global, discrete analysis of a collection of objects, as described by Guibas [20]. An example is determining whether a friendly vehicle is surrounded by a number of enemy tanks. Just as in the target counting problem, the ''am I surrounded'' relation can be resolved without having to solve the local problems of localizing all individual objects first.

10.5 Discussion

We have used the tracking problem as a vehicle to discuss sensor network CSIP design. We have focused on the estimation and tracking aspects and skipped over other important details, such as target detection and classification, for space reasons.

Detection is an important capability for a sensor network, as a tracker must rely on detection to initialize itself as new events emerge [1,22]. Traditional detection methods have focused on minimizing false alarms or the miss rate. In a distributed sensor network, the more challenging problem for detection is the proper allocation of sensing and communication resources to multiple competing detection tasks spawned by emerging stimuli. This dynamic allocation and focusing of resources in response to external events is somewhat analogous to attentional mechanisms in human vision systems, and is clearly a future research direction. More research should also be directed to the information architecture of distributed detection and tracking, and to addressing the problems of ''information double-counting'' and data association in a distributed network [6,23].

Optimizing resources for a given task, as for example in IDSQ, relies on accurate models of information gain and cost. To apply the information-driven approach to tracking problems involving other sensing modalities, or to problems other than tracking, we will need to generalize our models for sensing and estimation quality and our models of the tradeoff between resource use and quality. For example, what is the expected information gain per unit energy consumption in a network? One must make assumptions about the network, stimuli, and tasks in order to build such models. Another interesting problem for future research is to consider routing and sensing simultaneously and optimize for the overall gain of information.

We have not yet touched upon the programming issues in sensor networks. The complexity of the applications, the collaborative nature of the algorithms, and the plurality and diversity of resource constraints demand novel ways to construct, configure, test, and debug the system, especially the software. This is more challenging than traditional collection-based computation in parallel-processing research because sensor group management is typically dynamic and driven by physical events. In addition, the existing development and optimization techniques for embedded software are largely at the assembly level and do not scale to collaborative algorithms for large-scale distributed sensor networks. We need high-level system organizational principles, programming models, data structures, and processing primitives to express and reason about system properties, physical data, and their aggregation and abstraction, without losing relevant physical and resource constraints.

A possible programming methodology for distributed embedded sensing systems is shown in Figure 10.10. Given a specification at a collaborative behavioral level, software tools automatically generate the interactions of algorithm components and map them onto the physical hardware of sensor networks.

At the top level, the programming model should be expressive enough to describe application-level concerns, e.g. physical phenomena to be sensed, user interaction, and collaborative processing algorithms, without the need to manage node-level interactions. The programming model may be domain specific. For example, SAL [24] is a language for expressing and reasoning about geometries of physical data in distributed sensing and control applications; various biologically inspired computational models [25,26] study how complex collaborative behaviors can be built from simple

© 2005 by Chapman & Hall/CRC

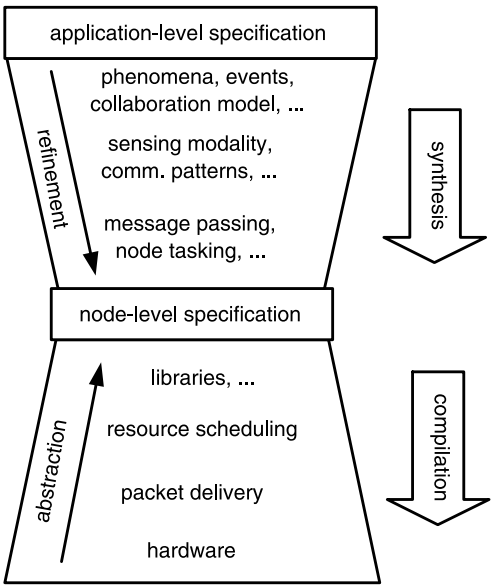


Figure 10.10. A programming methodology for deeply embedded systems.

components. The programming model should be structural enough to allow synthesis algorithms to exploit commonly occurring patterns and generate efficient code. TinyGALS [27] is an example of a synthesizable programming model for event-driven embedded software.

Automated software synthesis is a critical step in achieving the scalability of sensor network programming. Hardware-oriented concerns, such as timing and location, may be introduced gradually by refinement and configuration processes. The final outputs of software synthesis are operational code for each node, typically in forms of imperative languages, from which the more classical operating system, networking, and compiler technologies can be applied to produce executables. The libraries supporting node-level specifications need to abstract away hardware idiosyncrasy across different platforms, but still expose enough low-level features for applications to take advantage of.

10.6 Conclusion

This chapter has focused on the CSIP issues in designing and analyzing sensor network applications. In particular, we have used tracking as a canonical problem to expose important constraints in designing, scaling, and deploying these sensor networks, and have described approaches to several tracking problems that are at progressively higher levels with respect to the nature of information being extracted.

From the discussions, it is clear that, for resource-limited sensor networks, one must take a more holistic approach and break the traditional barrier between the application and networking layers. The challenge is to define the constraints from an application in a general way so that the networking layers can exploit, and *vice versa*. An important contribution of the approaches described in this chapter is the formulation of application requirements and network resources as a set of generic constraints so that target tracking and data routing can be jointly optimized.

Acknowledgments

This chapter was originally published in the August 2003 issue of the *Proceedings of the IEEE*. It is reprinted here with permission.

© 2005 by Chapman & Hall/CRC

The algorithm and experiment for the target counting problem were designed and carried out in collaboration with Qing Fang, Judy Liebman, and Elaine Cheong. The contour tracking algorithm and simulation were jointly developed with Krishnan Eswaran.

Patrich Cheung designed, prototyped, and calibrated the PARC sensor network testbeds and supported the laboratory and field experiments for the algorithms and software described in this chapter.

References

[1] Pottie, G.J. and Kaiser, W.J., Wireless integrated network sensors, *Communications of the ACM*, 43(5), 51, 2000.

[2] Intanagonwiwat, C. *et al.*, Directed diffusion: a scalable and robust communication paradigm for sensor networks, in *Proceedings of ACM MobiCOM*, Boston, August, 2000.

[3] Merrill, W.M. *et al.*, Open standard development platforms for distributed sensor networks, in *Proceedings of SPIE, Unattended Ground Sensor Technologies and Applications IV*, AeroSense 2002, Vol. 4743, Orlando, FL, April 2–5, 2002, 327.

[4] Hill, J. *et al.*, System architecture directions for networked sensors, in *ASPLOS 2000*.

[5] Doherty, L. *et al.*, Energy and performance considerations for smart dust, *International Journal of Parallel Distributed Systems and Networks*, 4(3), 121, 2001.

[6] Liu, J.J. *et al.*, Distributed group management for track initiation and maintenance in target localization applications, in *Proceedings of 2nd International Workshop on Information Processing in Sensor Networks (IPSN)*, April, 2003.

[7] Reid, D.B., An algorithm for tracking multiple targets, *IEEE Transactions on Automatic Control*, 24, 6, 1979.

[8] Bar-Shalom, Y. and Li, X.R., *Multitarget–Multisensor Tracking: Principles and Techniques*, YBS Publishing, Storrs, CT, 1995.

[9] Cox, I.J. and Hingorani, S.L., An efficient implementation of Reid's multiple hypothesis tracking algorithm and its evaluation for the purpose of visual tracking, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(2), 138, 1996.

[10] Poore, A.B., Multidimensional assignment formulation of data association problems arising from multitarget and multisensor tracking, *Computational Optimization and Applications*, 3, 27, 1994.

[11] Manyika, J. and Durrant-Whyte, H., *Data Fusion and Sensor Management: a Decentralized Information-Theoretic Approach*, Ellis Horwood, 1994.

[12] Estrin, D. *et al.*, Next century challenges: scalable coordination in sensor networks, in *Proceedings of the Fifth Annual International Conference on Mobile Computing and Networks (MobiCOM '99)*, Seattle, Washington, August, 1999.

[13] Brooks, R.R. *et al.*, Self-organized distributed sensor network entity tracking, *International Journal of High-Performance Computing Applications*, 16(3), 207, 2002.

[14] Zhao, F. *et al.*, Information-driven dynamic sensor collaboration, *IEEE Signal Processing Magazine*, 19(2), 61, 2002.

[15] Chu, M. *et al.*, Scalable information-driven sensor querying and routing for *ad hoc* heterogeneous sensor networks, *International Journal of High-Performance Computing Applications*, 16(3), 90, 2002.

[16] Liu, J.J. *et al.*, Multi-step information-directed sensor querying in distributed sensor networks, in *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, April, 2003.

[17] Liu, J.J. *et al.*, Collaborative in-network processing for target tracking, EURASIP *Journal of Applied Signal Processing*, 2003(4), 379, 2003.

[18] Fang, Q. *et al.*, Lightweight sensing and communication protocols for target enumeration and aggregation, in *ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2003.

© 2005 by Chapman & Hall/CRC

[19] Liu, J., *et al.*, A dual-space approach to tracking and sensor management in wireless sensor networks, in *Proceedings of 1st ACM International Workshop on Wireless Sensor Networks and Applications*, Atlanta, April, 2002, 131.
[20] Guibas L., Sensing, tracking, and reasoning with relations, *IEEE Signal Processing Magazine*, 19(2), 73, 2002.
[21] Tenney, R.R. and Sandell, N.R. Jr., Detection with distributed sensors, *IEEE Transactions on Aerospace and Electronic Systems*, 17, 501, 1981.
[22] Li, D. *et al.*, Detection, classification and tracking of targets in distributed sensor networks, *IEEE Signal Processing Magazine*, 19(2), 17, 2002.
[23] Shin, J. *et al.*, A distributed algorithm for managing multi-target identities in wireless *ad-hoc* sensor networks, in *Proceedings of 2nd International Workshop on Information Processing in Sensor Networks (IPSN)*, April, 2003.
[24] Zhao, F. *et al.*, Physics-based encapsulation in embedded software for distributed sensing and control applications, *Proceedings of the IEEE*, 91(1), 40, 2003.
[25] Abelson, H. *et al.*, Amorphous Computing, *Communications of the ACM*, 43(5), 74, 2001.
[26] Calude, C. *et al.* (eds), *Unconventional Models of Computation*, LNCS 2509, Springer, 2002.
[27] Cheong, E. *et al.*, TinyGALS: a programming model for event-driven embedded Systems, in *18th ACM Symposium on Applied Computing*, Melbourne, FL, March, 2003, 698.

© 2005 by Chapman & Hall/CRC

11

Environmental Effects

David C. Swanson

11.1 Introduction

Sensor networks can be significantly impacted by environmental effects from electromagnetic (EM) fields, temperature, humidity, background noise, obscuration, and for acoustic sensors outdoors, the effects of wind, turbulence, and temperature gradients. The prudent design strategy for intelligent sensors is to, at a minimum, characterize and measure the environmental effect on the sensor information reported while also using best practices to minimize any negative environmental effects. This approach can be seen as essential *information reporting* by sensors, rather than simple data reporting. Information reporting by sensors allows the data to be put into the context of the environmental and sensor system conditions, which translates into a confidence metric for proper use of the information. Reporting information, rather than data, is consistent with data fusion hierarchies and global situational awareness goals of the sensor network.

Sensor signals can be related to known signal patterns when the signal-to-noise ratio (SNR) is high, as well as when known environmental effects are occurring. A microprocessor is used to record and transmit the sensor signal, so it is a straightforward process to evaluate the sensor signal relative to known patterns. Does the signal fit the expected pattern? Are there also measured environmental parameters that indicate a possible bias or noise problem? Should the sensor measurement process adapt to this environmental condition? These questions can be incorporated into the sensor node's program flow chart to report the best possible sensor information, including any relevant environmental effects, or any unexplained environmental effect on SNR. But, in order to put these effects into an objective context, we first must establish a straightforward confidence metric based on statistical moments.

11.2 Sensor Statistical Confidence Metrics

Almost all sensors in use today have some performance impact from environmental factors that can be statistically measured. The most common environmental factors cited for electronic sensors are temperature and humidity, which can impact "error bars" for bias and/or random sensor error. If the environmental effect is a repeatable bias, then it can be removed by a calibration algorithm, such as in the corrections applied to J-type thermocouples as a function of temperature. If the bias is random from

© 2005 by Chapman & Hall/CRC

sensor to sensor (say due to manufacturing or age effects), then it can be removed via periodic calibration of each specific sensor in the network. But, if the measurement error is random due to low SNR or background noise interference, then we should measure the sensor signal statistically (mean and standard deviation). We should also report how the statistical estimate was measured in terms of numbers of observation samples and time intervals of observations. This also allows an estimate of the confidence of the variance to be reported using the Cramer–Rao lower bound [1]. If we assume the signal distribution has mean m and variance σ^2, and we observe N observations to estimate the mean and variance, our estimate has fundamental limitations on its accuracy. The estimated mean from N observations, m_N, is

$$m_N = m \pm \frac{\sigma}{\sqrt{N}} \tag{11.1}$$

The estimated variance σ_N^2 is

$$\sigma_N^2 = \sigma^2 \pm \frac{\sigma^2\sqrt{2}}{\sqrt{N}} \tag{11.2}$$

As Equations (11.1) and (11.2) show, only as N becomes large can the estimated mean and variance be assumed to match the actual mean and variance. For N observations, the mean and variance estimates cannot be more accurate than that seen in Equations (11.1) and (11.2) respectively. Reporting the estimated mean, standard deviation, the number of observations, and the associated time interval with those observations is essential to assembling the full picture of the sensor information.

For example, a temperature sensor's output in the absence of a daytime wind may not reflect air temperature only, but rather solar loading. To include this in the temperature information output one most have wind and insolation (solar heat flux) sensors and a physical algorithm to include these effects to remove bias in the temperature information. If the temperature is fluctuating, then it could be electrical noise or interference, but it could also be a real fluctuation due to air turbulence, partly cloudy skies during a sunny day, or cold rain drops. The temporal response of a temperature sensor provides a physical basis for classifying some fluctuations as electrical noise and others as possibly real atmospheric dynamics. For the electronic thermometer or relative humidity sensor, fluctuations faster than around 1 s can be seen as likely electrical noise. However, this is not necessarily the case for wind, barometer, and solar flux sensors.

11.3 Atmospheric Dynamics

The surface layer of the atmosphere is driven by the heat flux from the sun (and nighttime re-radiation into space), the latent heat contained in water vapor, the forces of gravity, and the forces of the prevailing geotropic wind. The physical details of the surface-layer are well described in an excellent introductory text by Stull [2]. Here, we describe the surface-layer dynamics as they pertain to unattended ground sensor (UGS) networks and the impact these atmospheric effects can have on acoustic, seismic, EM, and optical image data. However, we should also keep in mind that atmospheric parameters are physically interrelated and that the confidence and bias in a given sensor's reported data can be related physically to a calibration model with a broad range of environmental inputs.

Propagation outdoors can be categorized into four main wave types: acoustic, seismic, EM, and optical. While seismic propagation varies seasonally (as does underwater sound propagation), acoustic, optical, and EM wave propagation varies diurnally, and as fast as by the minute when one includes weather effects. The diurnal cycle starts with stable cold air near the ground in the early morning. As the sun heats the ground, the ground heats the air parcels into unstable thermal plumes, which rise upwards and draw cooler upper air parcels to the surface. The thermal plumes lead to turbulence and eventually to an increase in surface winds. Once the sun sets, the ground heating turns to radiation into space.

© 2005 by Chapman & Hall/CRC

The lack of solar heating stops the thermal plumes from forming. Colder parcels of air settle by gravity to the surface, forming a stable nocturnal boundary layer. The prevailing winds tend to be elevated over this stable cold air layer. Eventually, the cold air starts to drain downhill into the valleys and low-lying areas, in what are called katabatic winds. These nocturnal katabatic winds are very light and quite independent of the upper atmosphere prevailing geotropic winds. If there is a cold or warm front or storm moving across the area, then the wind and temperature tend to be very turbulent and fluctuating. It is important to keep in mind that these atmospheric effects on the surface are very local and have a significant effect on the ground sensor data. We will discuss these effects by wave type below.

11.3.1 Acoustic Environmental Effects

Acoustic waves outdoors have a great sensitivity to the local weather, especially wind. The sound wave speed in air is relatively slow at about 344 m/s at room temperature relative to wind speed, which can routinely approach tens of meters per second. The sound wave travels faster in downwind directions than upwind directions. Since the wind speed increases as one moves up in elevation (due to surface turbulence and drag), sound rays from a source on the ground tend to refract upwards in the upwind direction and downwards in the downwind direction. This means that a acoustic UGS will detect sound at much greater distances in the direction the wind is coming from. However, the wind will also generate acoustic noise from the environment and from the microphone. Wind noise is perhaps the most significant detection performance limitation for UGS networks. Turbulence in the atmosphere will tend to scatter and refract sound rays randomly. When the UGS is upwind of a sound source, this scattering will tend to make some of the upward refracting sound detectable by the UGS. The received spectra of the sound source in a turbulent atmosphere will fluctuate in amplitude randomly for each frequency due to the effects of multipaths. However, the effects of wind and turbulence on, and bearing measurement by, a UGS are usually quite small, since the UGS microphone array is typically only a meter or less. Building UGS arrays larger than about 2 m becomes mechanically complex, is subject to wind damage, and does not benefit from signal coherence and noise independence like EM or underwater arrays. This is because the sound speed is slow compared with the wind speed, making the acoustic signal spatially coherent over shorter distances in air.

When the wind is very light, temperature effects on sound waves tend to dominate. Sound travels faster in warmer air. During a sunny day the air temperature at the surface can be significantly warmer than that just a few meters above. On sunny days with light winds, the sound tends to refract upwards, making detection by a UGS more difficult. At night the opposite occurs, where colder air settles near the surface and the air above is warmer. Like the downwind propagation case, the higher elevation part of the wave outruns the slower wave near the ground. Thus, downward refraction occurs in all directions on a near windless night. This case makes detection of distant sources by a UGS much easier, especially because there is little wind noise. In general, detection ranges for a UGS at night can be two orders of magnitude better (yes, 100 times longer detection distances for the same source level). This performance characteristic is so significant that the UGS networks should be described operationally as nocturnal sensors. Figure 11.1 shows the measured sound from a controlled loudspeaker monitored continuously over a three day period.

Humidity effects on sound propagation are quite small, but still significant when considering long-range sound propagation. Water vapor in the air changes the molecular relaxation, thus affecting the energy absorption of sound into the atmosphere [3]. However, absorption is greatest in hot, dry air and at ultrasonic frequencies. For audible frequencies, absorption of sound by the atmosphere can be a few decibels per kilometer of propagation. When the air is saturated with water vapor, the relative humidity is 100% and typically fog forms from aerosols of water droplets. The saturation of water vapor in air h in grams per cubic meter can be approximated by

$$h\,(\mathrm{g/m^3}) = 0.94T + 0.345 \tag{11.3}$$

© 2005 by Chapman & Hall/CRC

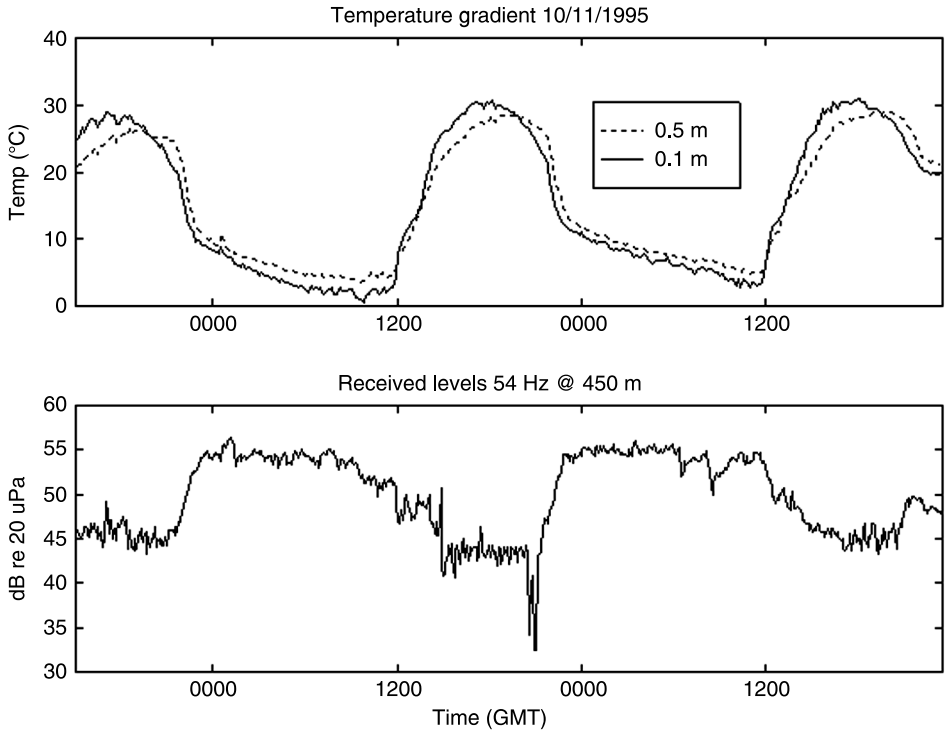


Figure 11.1. Air temperature at two heights (top) and received sound at 54 Hz from a controlled loudspeaker 450 m away over a 3 day period showing a 10 dB increase in sound during nighttime temperature inversions.

where T is in degrees centigrade. If the sensor reports the percent relative humidity RH at a given temperature T, one can estimate the dewpoint T_{dp}, or temperature where saturation occurs, by

$$T_{\mathrm{dp}} = \frac{(0.94T + 0.345) \times \mathrm{RH}/100 - 0.345}{0.94} \tag{11.4}$$

One can very simply approximate the saturation density h by the current temperature in centigrade and get the dewpoint temperature by multiplying by the relative humidity fraction. For example, if the temperature is 18°C there is roughly 18 g/m^3 of water vapor if the air is 100% saturated. If the RH is 20%, then there is roughly 3.6 g/m^3 of water vapor and the dewpoint is approximately 3.6°C. The actual number using Equations (11.3) and (11.4) are 17.3 g/m^3 for saturation, 3.46 g/m^3 for 20% RH, and a dewpoint of 3.3°C. Knowledge of the humidity is useful in particular for chemical or biological aerosol hazards, EM propagation, and optical propagation, but it does not significantly affect sound propagation. Humidity sensors generally are only accurate to a few percent, unless they are the expensive "chilled mirror" type of optical humidity sensor. There are also more detailed relative humidity models for estimating dewpoint and frost-point temperatures in the literature. Equations (11.3) and (11.4) are a useful and practical approximation for UGS networks.

11.3.2 Seismic Environmental Effects

Seismic waves are relatively immune to the weather, except for cases where ground water changes or freezes. However, seismic propagation is significantly dependent on the subterranean rock structures and material. Solid materials, such as a dry lake bed, form ideal seismic wave propagation areas. Rock fissures, water, and back-filled areas of earth tend to block seismic waves. If one knows the seismic

© 2005 by Chapman & Hall/CRC

propagation details for a given area, then seismic arrays can make very effective UGS networks. If the source is on or near the surface, two types of wave are typically generated, a spherically radiating pressure wave, or p-wave, and a circularly radiating surface shear wave, or s-wave. The p-wave can carry a lot of energy at very fast speeds due to the compressional stiffness of the ground. However, since it is spherically spreading (approximately) its amplitude (dB) decays with distance R by $20 \log R$, or 60 dB in the first 1 km.

The s-wave speed depends on the shear stiffness of the ground and the frequency, where high frequencies travel faster than low frequencies. Since the s-wave spreads circularly on the surface, its approximate amplitude dependence (dB) with distance R is $10 \log R$, or only 30 dB in the first 1 km. A UGS network detecting seismic waves from ground vehicles is predominately detecting s-waves, the propagation of which is highly dependent on the ground structure. In addition, there will always be a narrow frequency range where the s-wave speed is very close to the acoustic wave speed in the air. This band in the seismic spectrum will detect acoustic sources as well as seismic sources. Seismic sensors (typically geophones) will also detect wind noise through tree roots and structure foundations. If the UGS sensor is near a surf zone, rapids, airport, highway, or railroad, it will also detect these sources of noise or signal, depending on the UGS application. When the ground freezes, frozen water will tend to make the surface stiffer and the s-waves faster. Snow cover will tend to insolate the ground from wind noise. Changes in soil moisture can also effect seismic propagation, but in complicated ways depending on the composition of the soil.

11.3.3 EM Environmental Effects

Environmental effects on EM waves include the effect of the sun's radiation, the ionosphere, and, most important to UGS communications, the humidity and moisture on the ground. The water vapor density in the air, if not uniform, has the effect of changing the EM impedance, which can refract EM waves. When the air is saturated, condensation in the form of aerosols can also weaken EM wave propagation through scattering, although this effect is fairly small at frequencies below 60 GHz (the wavelength at 60 GHz is 5 mm). In the 100s of Megahertz range, propagation is basically line of sight except for the first couple of reflections from large objects, such as buildings (the wavelength at 300 MHz is 1 m). Below 1 MHz, the charged particles of the Earth's ionosphere begin to play a significant role in the EM wave propagation. The ground and the ionosphere create a waveguide, allowing long-range ''over the horizon'' wave propagation. The wavelength at 300 kHz is 1 km, so there are little environmental effects from manmade objects in the propagation path, provided one has a large enough antenna to radiate such a long wavelength. In addition, EM radiation by the sun raises background noise during the daytime.

For all ground-to-ground EM waves, the problem for UGS networks sending and receiving is the practical fact that the antenna needs to be small and cannot have a significant height above the ground plane. This propagation problem is crippling when the dewpoint temperature (or frost point) is reached, which effectively can raise the ground plane well above a practical UGS antenna height, rendering the antenna efficiency to minimal levels. Unfortunately, there are no answers for communication attempts using small antennas near the ground in high humidity. Vegetation, rough terrain, limited line of sight, and especially dew- or frost-covered environments are the design weak point of UGS networks. This can be practically managed by vertical propagation to satellites or air vehicles. Knowledge of humidity and temperature can be very useful in assessing the required power levels for EM transmission, as well as for managing communications during problem environmental conditions.

11.3.4 Optical Environmental Effects

Optical waves are also EM waves, but with wavelengths in the 0.001 mm (infrared) to a few hundred nanometers. The visual range is from around 700 nm (red) to 400 nm (violet) wavelengths. These small

© 2005 by Chapman & Hall/CRC

wavelengths are affected by dust particles, and even the molecular absorption by the atmospheric gases. Scattering is stronger at shorter wavelengths, which is why the sky is blue during the day. At sunrise and sunset, the sunlight reaches us by passing through more of the atmosphere, this scattering that blue light, leaving red, orange, and yellow. Large amounts of pollutants, such as smoke, ozone, hydrocarbons, and sulfur dioxide, can also absorb and scatter light, obscuring optical image quality. Another obvious environmental effect for imagery is obscuration by rain or snow. However, thermal plumes and temperature gradients also cause local changes in air density and the EM index of refraction, which cause fluctuations in images.

Measuring the environmental effects directly can provide an information context for image features and automatic target recognition. The syntax for logically discounting or enhancing the weight of some features in response to the environment creates a very sophisticated and environmentally robust UGS. More importantly, it provides a scientific strategy for controlling false alarms due to environmental effects.

11.3.5 Environmental Effects on Chemical and Biological Detection and Plumes Tracking

Perhaps one of the most challenging and valuable tasks of a UGS network on the battlefield is to provide real-time guidance on the detection and tracking of plumes of harmful chemical vapors, aerosols, or biological weapons. The environment in general, and in particular the temperature and humidity, have an unfortunate direct effect on the performance of many chemical and biological sensors. Putting the chemical and biological sensor performance aside, the movement and dispersion of a detected chem/bio plume is of immediate importance once it is detected; thus, this capability is a major added value for UGS networks.

Liquid chemical and biological aerosols will evaporate based on their vapor pressures at a particular temperature and the partial pressures of the other gases, most notably water, in the atmosphere. Once vaporized, the chemicals will diffuse at nearly the speed of sound and the concentration will decrease rapidly. Since vaporization and diffusion are highly dependent on temperature, the local environmental conditions play a dominant role in how fast a chemical threat will diffuse and which direction the threat will move. To maximize the threat of a chemical weapon, one would design the material to be a power of low vaporization aerosol to maintain high concentration for as long as possible in a given area [4]. The environmental condition most threatening to people is when a chemical or biological weapon is deployed in a cold, wet fog or drizzle where little wind or rain is present to help disperse the threat. During such conditions, temperature inversions are present where cold stable air masses remain at the surface, maximizing exposure to the threat. A UGS network can measure simple parameters such as temperature gradients, humidity, and wind to form physical features, such as the bulk Richardson index R_B, to indicate in a single number the stability of the atmosphere and the probability of turbulence, as seen in Equation (11.5) and Figure 11.2 [5].

$$R_B = \frac{g\Delta T \Delta z}{T(\Delta U^2 + \Delta V^2)} \tag{11.5}$$

In Equation (11.5), the parameters ΔU and ΔV represent the horizontal wind gradient components, g is the acceleration due to gravity, ΔT is the temperature gradient (top minus bottom), and Δz is the separation of the temperature readings. If the bottom is the ground surface, then one can assume that the wind is zero. Clearly, this is a good example of a natural physical feature for the state of the environment. When $R_B > 1$, stable air near the ground enhances the exposure of chemical or biological weapons. This problem arises from the lack of turbulence and mixing that helps disperse aerosols.

For measurements such as wind, it not only makes sense to report the mean wind speed, but also the standard deviation, sample interval, and number of samples in the estimate. Information on the sample

© 2005 by Chapman & Hall/CRC

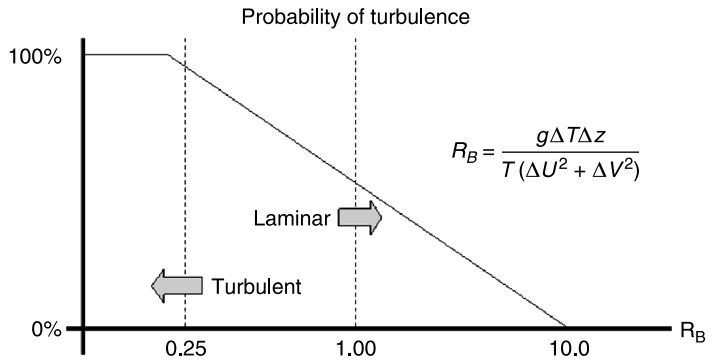


Figure 11.2. The bulk Richardson index provides a physical atmospheric parameter representing the likelihood of turbulence given the wind and temperature measured by a UGS.

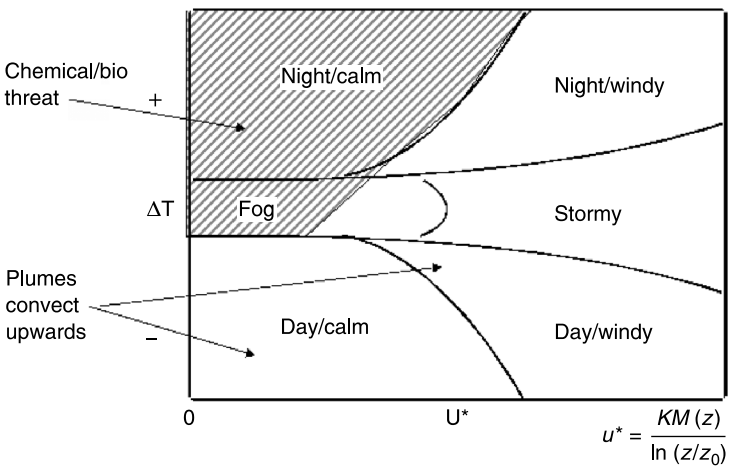


Figure 11.3. Using temperature gradient and wind M, we can show a simplified atmospheric state space that illustrates the conditions for an elevated chemical or biological threat ($k = 0.4$).

set size is used in Equations (11.1) and (11.2) to provide information confidence bounds, rather than simply data. Combining wind and temperature data we can devise a chart for atmospheric state, as seen in Figure 11.3.

A UGS can extract another important turbulence factor from the wind called the dissipation rate of the Kolmogorov spectrum [2]. The Kolmogorov spectrum represents the wave structure of the turbulence. It is useful for sound propagation models and for chemical and biological plume transport models. One calculates the mean wind speed and the Fourier transform of a regularly time-sampled series of wind-speed measurements. The mean wind speed is used to convert the time samples to special samples such that the Fourier spectrum represents a wavenumber spectrum. This is consistent with Taylor's hypothesis [2] of spatially frozen turbulence, meaning that the spatial turbulence structure remains essentially the same as it drifts with the wind. Figure 11.4 shows the Kolmogorov spectrum for over 21 h of wind measurements every 5 min. The physical model represented by the Kolmogorov spectrum has importance in meteorology, as well as in sound-propagation modeling, where turbulence cause random variations in sound speed.

The surface layer of the atmosphere is of great interest to the meteorologist because much of the heat energy transport occurs there. There is the obvious heating by the sun and cooling by the night sky,

© 2005 by Chapman & Hall/CRC

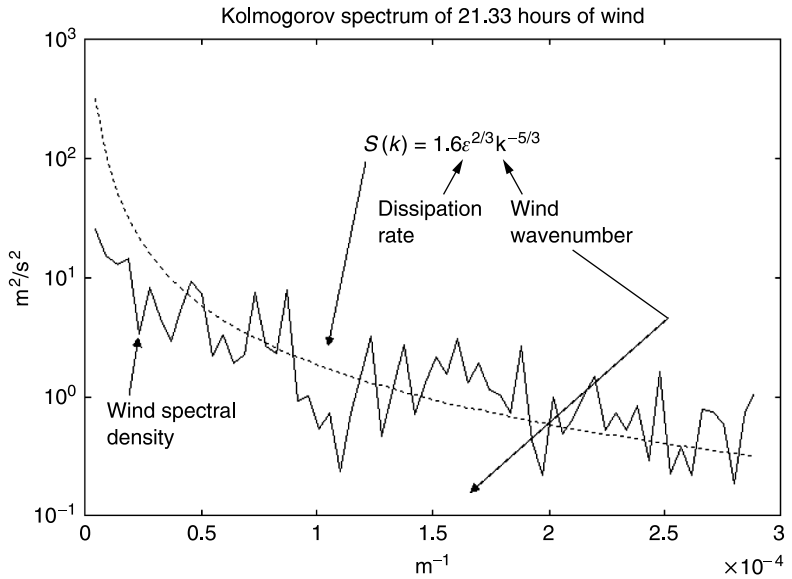


Figure 11.4. The Kolmogorov spectrum provides a means to characterize the turbulent structure of the wind using a single parameter called the dissipation rate ε.

both of which are significantly impacted by moisture and wind. There is also a latent heat flux associated with water vapor given off by vegetation, the soil, and sources of water. While atmospheric predictive models such as MM5 can provide multi-elevation weather data at grid points as close as 20 km, having actual measurements on the surface is always of some value, especially if one is most interested in the weather in the immediate vicinity. These local surface inputs to the large-scale weather models are automated using many fixed sites and mobile sensors (such as on freight trucks) that provide both local and national surface-layer measurements.

11.4 Propagation of Sound Waves

Sound will travel faster in downwind directions and in hotter air, whereas aerosols will disperse slowly in the still cool air of a nocturnal boundary layer. The atmospheric condition of concern for both noise and air pollution is the case of a cool, still, surface boundary layer, as this traps pollutants and downward-refracts sound waves near the ground and often occurs at night. The local terrain is very important to the formation of a nocturnal boundary layer, as low-lying and riverine areas will tend to collect cold parcels of heavy air. Slow katabatic winds form from the ''draining'' of these cool air parcels downhill. Local sensors can detect the environmental conditions leading to the formation of nocturnal boundary layers in these local areas to help avoid problems from local noise and chemical pollution.

Given the weather information, we can construct a volume of elements each with a sound speed plus a wind vector. The wind speed adds to the sound speed in the direction of propagation, which is calculated as a dot product of the wind vector and the propagation direction vector, plus the sound speed scalar. We will develop a general propagation algorithm by first starting with a simple point source (small compared with wavelength) and noting the pressure field created:

$$p(r, k) = \frac{A}{R} e^{j(\omega t - kr)} \tag{11.6}$$

The pressure p in Equation (11.6) decays with distance r for frequency ω and wavenumber k. Equation (11.6) describes an outgoing wave; as can be seen, as the time t increases, so the distance r must also increase to keep the phase constant. Since we are modeling wave propagation in one direction only,

© 2005 by Chapman & Hall/CRC

we can adopt a cylindrical coordinate system and concern our model with a particular direction only. However, this requires that we factor a square root of distance out to account for cylindrical verse spherical spreading.

$$\psi_{2D}(r,\, k) = \sqrt{r} p_{3D}(r, k) \tag{11.7}$$

We can now decompose the wavenumber k into its r (distance) and z (elevation) components. Since $k = \omega/c$, where ω is radian frequency and c is the sound speed, the meteorological variations in sound speed will impact the wavenumber:

$$k = \sqrt{k_r^2 + k_z^2} \tag{11.8}$$

The pressure wavenumber spectrum of the field at some distance r is the Fourier transform of a slice of the field along the z-direction:

$$\psi(r, k_z) = \int_{-\infty}^{+\infty} \psi(r, z)\, \mathrm{e}^{-\mathrm{j}k_z z}\, \mathrm{d}z \tag{11.9}$$

Using wavenumber spectra, we can write the spectrum at a distance $r + \Delta r$in terms of the spectrum at r:

$$\psi(r + \Delta r, k_z) = \mathrm{e}^{\mathrm{j}\Delta r\sqrt{k^2 - k_z^2}} \psi(r, k_z) \tag{11.10}$$

Equation (11.10) may not seem significant, but one cannot relate the pressure fields at r and $r + \Delta r$ directly in this manner.

We now consider variations in the total wavenumber k, due to small variations in sound speed, as the horizontal wavenumber at the surface $k_r(0)$ plus a small variation in wavenumber due to the environmental sound speed:

$$k^2 = k_r^2(0) + \delta k^2(z) \tag{11.11}$$

If the wavenumber variation is small, say on the order of a few percent or less, k_r can be approximated by

$$k_r \approx \sqrt{k_r^2(0) - k_z^2(r)} + \frac{\delta k^2(r,\, z)}{2k_r(0)} \tag{11.12}$$

Equation (11.10) is now rewritten as

$$\psi(r + \Delta r, k_z) = \mathrm{e}^{\mathrm{j}\Delta r \frac{\delta k^2(r,z)}{2k_r(0)}}\ \mathrm{e}^{\mathrm{j}\Delta r\sqrt{k_r^2(0) - k_z^2(r)}} \psi(r, k_z) \tag{11.13}$$

and using Fourier transforms can be seen as

$$\psi(r + \Delta r, z) = \mathrm{e}^{\mathrm{j}\Delta r \frac{\delta k^2(r,z)}{2k_r(0)}} \frac{1}{2\pi} \int_{-\infty}^{+\infty} \psi(r, k_z)\ \mathrm{e}^{\mathrm{j}\Delta r\sqrt{k_r^2(0) - k_z^2(r)}} \mathrm{e}^{\mathrm{j}k_z z}\, \mathrm{d}z \tag{11.14}$$

so that we have a process cycle of calculating a Fourier transform of the acoustic pressure, multiplying by a low-pass filter, inverse Fourier transforming, and finally multiplying the result by a phase variation

© 2005 by Chapman & Hall/CRC

with height for the particular range step. In this process, the acoustic wave fronts will diffract according to the sound speed variations and we can efficiently calculate good results along the r-direction of propagation.

For real outdoor sound propagation environments, we must also include the effects of the ground. Following the developments of Gilbert and Di [6], we include a normalized ground impedance $Z_g(r)$ with respect to $\rho c = 415$ Rayls for the impedance of air. This requires a ground reflection factor

$$R(k_z) = \frac{k_z(r)Z_g(r) - k_r(0)}{k_z(r)Z_g(r) + k_r(0)} \tag{11.15}$$

and a surface complex wavenumber $\beta = k_r(0)/Z_g(r)$, which accounts for soft grounds.

The complete solution is

$$\psi(r + \Delta r, z) = e^{j\Delta r \frac{gk^2(r,z)}{2k_r(0)}} \left[\begin{array}{l} \left\{ \dfrac{1}{2\pi} \displaystyle\int_{-\infty}^{+\infty} \left[\psi(r, k_z)\ e^{j\Delta r \sqrt{k_r^2(0) - k_z^2(r)}} \right] e^{+jk_z z}\, dz \right\} \\ \quad + \left\{ \dfrac{1}{2\pi} \displaystyle\int_{-\infty}^{+\infty} \left[\psi(r, -k_z) R(k_z)\ e^{j\Delta r \sqrt{k_r^2(0) - k_z^2(r)}} \right] e^{+jk_z z}\, dz \right\} \\ \quad + 2j\beta e^{-j\beta(r)z}\ e^{j\Delta r \sqrt{k_r^2(0) - \beta^2(r)}} \psi(r, \beta) \end{array} \right] \tag{11.16}$$

where the first term in Equation (11.16) is the direct wave, the middle term is the reflected wave, and the last term is the surface wave. For rough surfaces, one can step up or down the elevation as a phase shift in the wavenumber domain. Figure 11.5 shows some interesting results using a Green's function parabolic equation model.

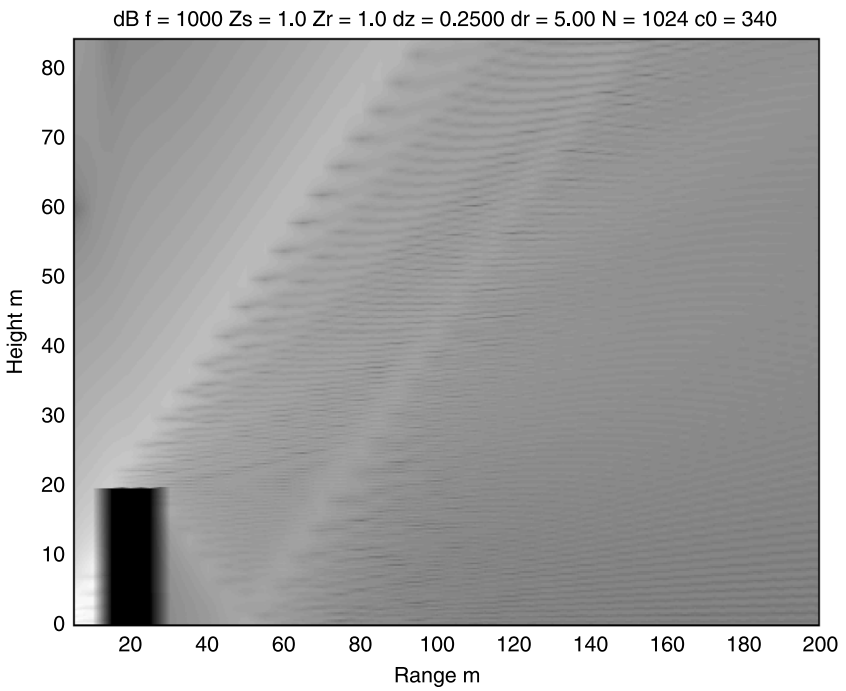


Figure 11.5. Propagation model results with a source to the left of a large wall showing diffraction, ground reflection, and atmospheric refraction.

© 2005 by Chapman & Hall/CRC

References

[1] Swanson, D.C., *Signal Processing for Intelligent Sensor Systems,* Marcel-Dekker, New York, 2000, 362.
[2] Stull, R., *An Introduction to Boundary Layer Meteorology*, Kluwer Academic Publishers, 1991.
[3] Kinsler, L.E. and Frey, A.R., *Fundamentals of Acoustics,* Wiley, New York, 1962, 227.
[4] Ali, J. *et al*., *US Chemical–Biological Defense Guidebook,* Jane's Information Group, Alexandria, 1997, 150.
[5] Garratt, J.R., *The Atmospheric Boundary Layer,* Cambridge University Press, New York, 1992, 32.
[6] Gilbert, K.E. and Di, X., A fast Green's function method for one-way sound propagation in the atmosphere, *Journal of the Acoustical Society of America,* 94(4), 2343, 1993.

© 2005 by Chapman & Hall/CRC

12

Detecting and Counteracting Atmospheric Effects

Lynne L. Grewe

12.1 Motivation: The Problem

With the increasing use of vision systems in uncontrolled environments, reducing the effect of atmospheric conditions (such as fog) in images has become a significant problem. Many applications (e.g. surveillance) rely on accurate images of the objects under scrutiny. Poor visibility is an issue in aircraft navigation [1–4], highway monitoring [5], commercial and military vehicles [6,7]. Figure 12.1 shows an example scene from a visible spectrum camera where the presence of fog severely impedes the ability to recognize objects in the scene.

Before discussing how particular sensors respond to atmospheric conditions and algorithms to improve the resulting images, let us discuss what is meant by atmosphere.

12.1.1 What Is Atmosphere?

Atmosphere, whether caused by fog, rain or even smoke, involves the presence of particles in the air. On a clear day, the particles present which make up "air," like oxygen, are so small that sensors are not impaired in their capture of the scene elements. However, this is not true with the presence of atmospheric conditions like fog, because these particles are larger in size and impede the transmission of light (electromagnetic radiation) from the object to the sensor through scattering and absorption.

There have been many models proposed to understand how these particles interact with electromagnetic radiation. These models are referred to as scattering theories. Models consider parameters like radius of particle size, wavelength of light, density of particle material, shape of particles, etc. Selection of the appropriate model is a function of the ratio of the particle radius to the wavelength of light being considered, as well as what parameters are known.

When this ratio of particle size to wavelength is near one, most theories used are derived from Mie scattering theory [8]. This theory is the result of solving the Maxwell equations for the interaction of an electromagnetic wave with a spherical particle. This theory takes into account absorption and the refractive index (related to the angle through which light is bent when encountering a particle).

© 2005 by Chapman & Hall/CRC



Figure 12.1. Building in foggy conditions; visibility of objects is impaired.

Other theories exist that do not assume the particle has a spherical shape and alter the relationship between size, distribution, reflection, and refraction. The sizes and shapes of these particles vary. For example, water-based particles range from some micrometers and perfect spheres in the case of liquid cloud droplets to large raindrops (up to 9 mm diameter), which are known to be highly distorted. Ice particles have a variety of nonspherical shapes. Their size can be some millimeters in the case of hail and snow, but also extends to the regime of cirrus particles, forming needles or plates of ten to hundreds of micrometers.

Most scattering theories use the following basic formula that relates the incident (incoming) spectral irradiance $E(\lambda)$ to the outgoing radiance in the direction of the viewer $I(\theta, \lambda)$ and the angular scattering function $\beta(\theta, \lambda)$.

$$I(\theta, \lambda) = \beta(\theta, \lambda)E(\lambda) \tag{12.1}$$

where λ is the wavelength of the light and θ is the angle the viewer is at with regard to the normal to the surface of the atmospheric patch the incident light is hitting. What is different among the various models is $\beta(\theta, \lambda)$.

An observed phenomenon regarding physics-based models is that light energy at most wavelengths λ is attenuated as it travels through the atmosphere. In general, as it travels through more of the atmosphere, the signal will be diminished as shown in Figure 12.2. This is why, as humans, we can only see objects close to us in heavy fog. This is described as the change in the incoming light as follows (from [10]):

$$\frac{\mathrm{d}E(x, \lambda)}{E(x, \lambda)} = -\beta(\lambda)\,\mathrm{d}x \tag{12.2}$$

where x is the travel direction and $\mathrm{d}x$ is the distance travelled along that direction. Note that $\beta(\lambda)$, called the total scattering function, is simply $\beta(\theta, \lambda)$ integrated over all angles. If we travel from the starting point of $x = 0$ to a distance d, then the following will be the irradiance at d:

$$E(d, \lambda) = -E_{\mathrm{o}}(\lambda)\exp(-\beta(\lambda)d) \tag{12.3}$$

where $E_{\mathrm{o}}(\lambda)$ is the initial irradiance at $x = 0$.

© 2005 by Chapman & Hall/CRC

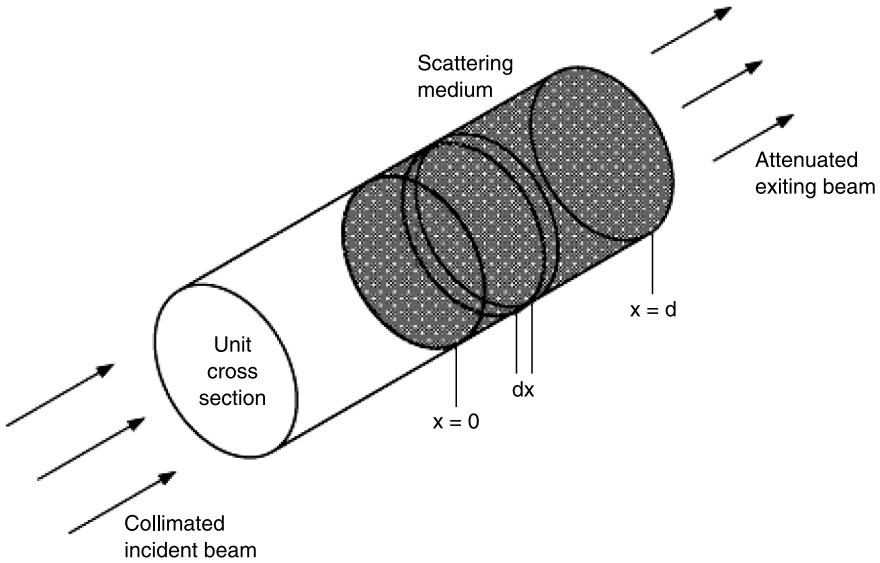


Figure 12.2. Demonstration of light attenuating as it travels through the atmosphere (from [9]).

Many of the scattering theories treat each particle as independent of other particles because their separation distance is many times greater than their own diameter. However, multiple scatterings from one particle to the next of the original irradiance occur. This phenomenon, when produced from environmental illumination like direct sunlight, was coined "airlight" by Koschmieder [11]. In this case, unlike the previously described attenuation phenomenon, there is an increase in the radiation as the particles inter-scatter the energy. Humans experience this as foggy areas being "white" or "bright". Figure 12.3, from Narasimhan and Nayar [9], illustrates this process. Narasimhan and Nayar [9] developed a model that integrates the scattering equation of Equation (12.1) with the airlight phenomenon using an overcast environmental illumination model present on most foggy days.

The reader is referred to Curry *et al.* [12], Day *et al.* [13], Kyle [14] and Bohren [15] for details on specific atmospheric scattering theories and their equations.

12.2 Sensor-Specific Issues

Distributed sensor networks employ many kinds of sensor. Each kind of sensor responds differently to the presence of atmosphere. While some sensors operate much better than others in such conditions, it is true that in most cases there will be some degradation to the image. We will highlight a few types of sensor and discuss how they are affected.

The system objectives, including the environmental operating conditions, should select the sensors used in a sensor network. An interesting study that compares the response of visible, near-IR, and thermal-IR sensors in terms of target detection is that of Sadot *et al.* [16].

12.2.1 Visible-Spectrum Cameras

Visual-spectrum photometric images are very susceptible to atmospheric effects. As shown in Figure 12.1, the presence of atmosphere causes the image quality to degrade. Specifically, images can appear fuzzy, out of focus, objects may be obscured behind the visual blanket of the atmosphere, and other artifacts may occur. As discussed in the previous section, images appear brighter in fog and haze, with the atmosphere itself often being near-white, and farther scene objects are not bright or are fuzzy.

© 2005 by Chapman & Hall/CRC

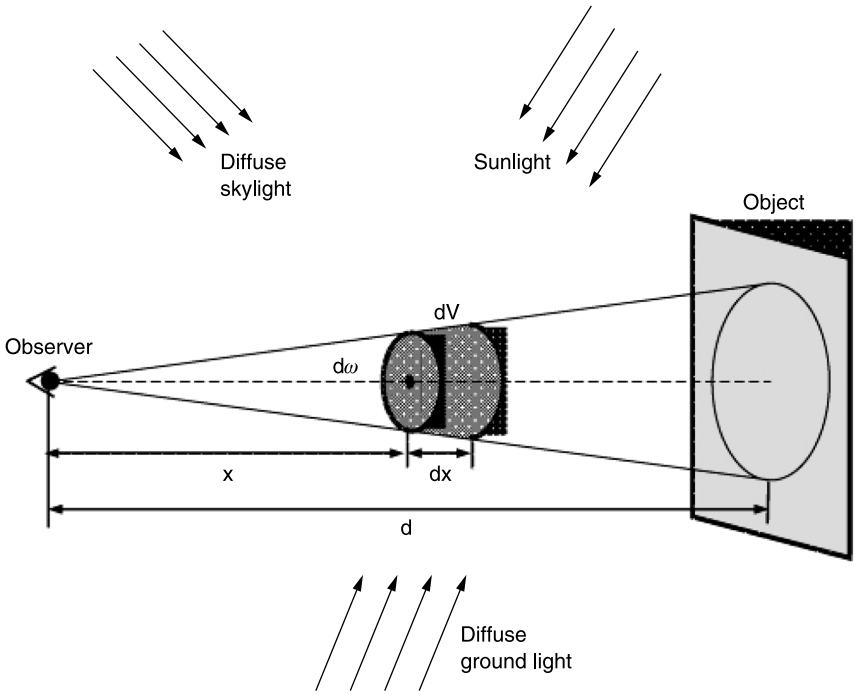


Figure 12.3. Cone of atmosphere between an observer and object scatters environmental illumination in the direction of the observer. This, acts like a light source, called an "airlight," the brightness of which increases with path length *d*. From [9].

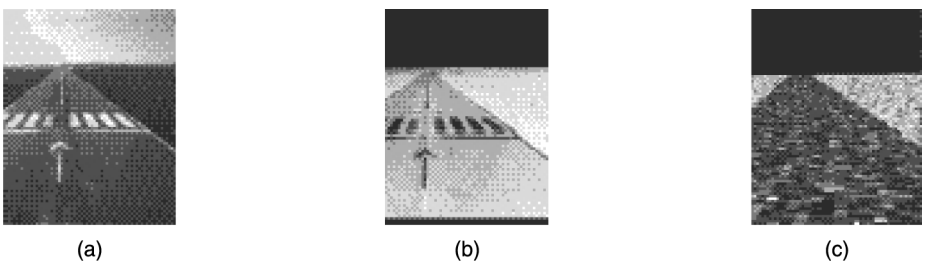


Figure 12.4. Different sensor images of the same scene, an airport runway: (a) visible spectrum; (b) infrared (IR); (c) millimeter wave (MMW).

Figure 12.4 shows a visible-spectrum image and the corresponding IR and MMW images of the same scene. Figure 12.5 shows another visible-spectrum image in foggy conditions.

12.2.2 IR Sensors

IR light can be absorbed by water-based molecules in the atmosphere, but imaging using IR sensors yields better results than visible-spectrum cameras. This is demonstrated in Figure 12.5. As a consequence, IR sensors have been used for imaging in smoke and adverse atmospheric conditions. For example, IR cameras are used by firefighters to find people and animals in smoke-filled buildings. IR, as discussed in Chapter 6, is often associated with thermal imaging and is used in night-vision systems. Of course some "cold" objects, which do not put out IR energy, cannot be sensed by an IR sensor and hence in comparison to a visible spectrum image of the scene may lack information or

© 2005 by Chapman & Hall/CRC



Figure 12.5. IR is better in imaging through fog than a visible-spectrum camera: (a) visible-spectrum image on clear day; (b) grayscale visible spectrum image on a foggy day; (c) IR image on the same foggy day; (d) visible and IR images of a runway. (a)–(c) From Hosgood [17], (d) from NASA [18].

desired detail as well as introduce unwanted detail. This lack of detail can be observed by comparing Figure 12.5(a) and (c) and noting that the IR image would not change much in clear conditions. Figure 12.11, below, also shows another pair of IR and visible-spectrum images.

12.2.3 MMW Radar Sensors

MMW radar is another good sensor for imaging in atmospheric conditions. One reason is that, compared with many sensors, it can sense at a relatively long range. MMW radar works by emitting a beam of electromagnetic waves. The beam is scanned over the scene and the reflected intensity of radiation is recorded as a function of return time. The return time is correlated with range, and this information is used to create a range image.

In MMW radar, in comparison with many other sensors, the frequency of the beam allows it to pass by atmospheric particles because they are too small to affect the beam. MMW radar can thus "penetrate" atmospheric layers like smoke, fog, etc. However, because of the frequency, the resolution of the image produced, compared with many other sensors, is poorer. This can be Figure 12.4(c), which shows the MMW radar image produced alongside the photometric and IR images of the same scene. In addition, MMW is measuring range, and not reflected color or other parts of the spectrum that may be required for the task at hand.

12.2.4 LADAR Sensors

Another kind of radar system is that of laser radar, or LADAR. LADAR sensors use shorter wavelengths than other radar systems (e.g. MMW) and can thus achieve better resolution. Depending on the application, this resolution may be a requirement. A LADAR sensor sends out a laser beam and the time of return in the reflection measures the distance from the scene point. Through scanning of the scene, a range image is created. As the laser beam propagates through the atmosphere, fog droplets and raindrops cause image degradation, which is manifested as either dropouts (meaning not enough reflection is returned to be registered) or false returns (the beam is returned from the atmosphere particle itself). Campbell [19] studied this phenomenon for various weather conditions, yielding a number of performance plots. One conclusion was that, for false returns to occur, the atmospheric moisture droplets (rain) had to be at least 3 mm in diameter. This work was done with a 1.06 μm wavelength LADAR sensor. The results would change with change in wavelength. The performance plots produced can be used as thresholds in determining the predicted performance of the system given current atmospheric conditions and could potentially be used to select image-processing algorithms to improve image quality.

12.2.5 Multispectral Sensors

Satellite imagery systems often encounter problems with analysis and clarity of original images due to atmospheric conditions. Satellites must penetrate through many layers of the Earth's atmosphere in

© 2005 by Chapman & Hall/CRC

order to view ground or near-ground scenes. Hence, this problem has been given great attention in research. Most of this work involves various remote-sensing applications, like terrain mapping, vegetation monitoring, and weather prediction and monitoring systems. However, with high spatial resolution satellites, other less ''remote'' applications, like surveillance and detailed mapping, can suffer from atmospheric effects.

12.3 Physics-Based Solutions

As discussed earlier, physics-based algorithms is one of the two classifications that atmosphere detection and correction systems can be grouped into. In this section, we describe a few of the systems that use physics-based models. Most of these systems consider only one kind of sensor. Hence, in heterogeneous distributed sensor networks one would have to apply the appropriate technique for each different kind of sensor used. The reader should read Section 12.1.1 for a discussion on atmosphere and physics-based scattering before reading this section.

Narasimhan and Nayar [9] discuss in detail the issue of atmosphere and visible-spectrum images. They developed a dichromatic atmosphere scattering model that tracks how images are affected by atmosphere particles as a function of color, distance, and environmental lighting interactions (''airlight'' phenomenon; see Section 12.1.1). In general, when particle sizes are comparable to the wavelength of a reflected object's light, the transmitted light through the particle will have its spectral composition altered. For fog and dense haze, they suggest that the shifts in the spectral composition for the visible spectrum are minimal, and hence they assume that the hue of the transmitted light is independent of the depth of the atmosphere. They postulate that, with an overcast sky, the hue of airlight depends on the particle size distribution and tends to be gray or light blue in the case of haze and fog. Recall that airlight is the transmitted light that originated directly from environmental light like direct sunlight. Their model for the spectral distribution of light received by the observer is the sum of the distribution from the scene objects' reflected light taking into account the attenuation phenomenon (see Section 12.1.1) and airlight. It is similar to the dichromatic reflectance model of Shafer [20], which describes the spectral effects of diffuse and specular surface reflections.

Narasimhan and Nayar [9] reduce their wavelength-based equations to the red, green and blue color space, and in doing so then show that the equation relating to the received color remains a linear combination of the scene object transmitted light color and the environmental airlight color. Narasimhan and Nayar [9] hypothesize that a simple way of reducing the effect of atmosphere on an image would be to subtract the airlight color component from the image.

Hence, Narasimhan and Nayar [9] discuss how to measure the airlight color component. A color component (like airlight) is described by a color unit vector and its magnitude. The color unit vector for airlight can be estimated as the unit vector of the average color in an area of the image that should be registered as black. However, this kind of calibration may not be possible, and they discuss a method of computing it using all of the color pixel values in an image formulated as an optimization problem.

Using the estimate of the airlight color component's unit vector, and given the magnitude of this component at just one point in an image, as well as two perfectly registered images of the scene, Narasimhan and Nayar [9] are able to calculate the airlight component at each pixel. Subtracting the airlight color component at each pixel value yields an atmosphere-corrected image. It is also sufficient to know the true transmission color component at one point to perform this process. Figure 12.6 shows the results of this process. This system requires multiple registered images of the scene as well as true color information at one pixel in the image. If this is not possible, then a different technique should be applied. Obviously, this technique works for visible-spectrum images and under the assumption that the atmosphere condition does not (significantly) alter the color of the received light.

A method for correcting satellite imagery taken over mountainous terrain has been developed by Richter [21] to remove atmospheric and topographic effects. The algorithm accounts for horizontally varying atmospheric conditions and also includes the height dependence of the atmospheric radiance and transmittance functions to simulate the simplified properties of a three-dimensional atmosphere.

© 2005 by Chapman & Hall/CRC

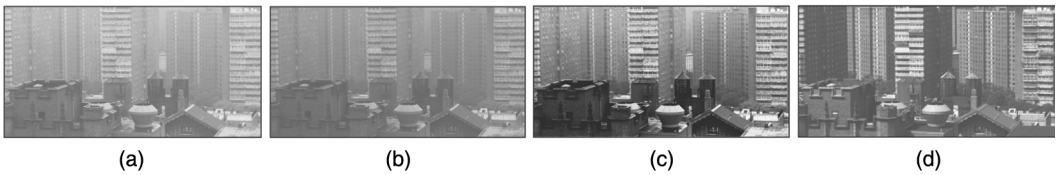


Figure 12.6. Narasimhan and Nayar [9] fog removal system: (a) and (b) foggy images under overcast day; (c) defogged image; (d) image taken on clear day under partly cloudy sky.

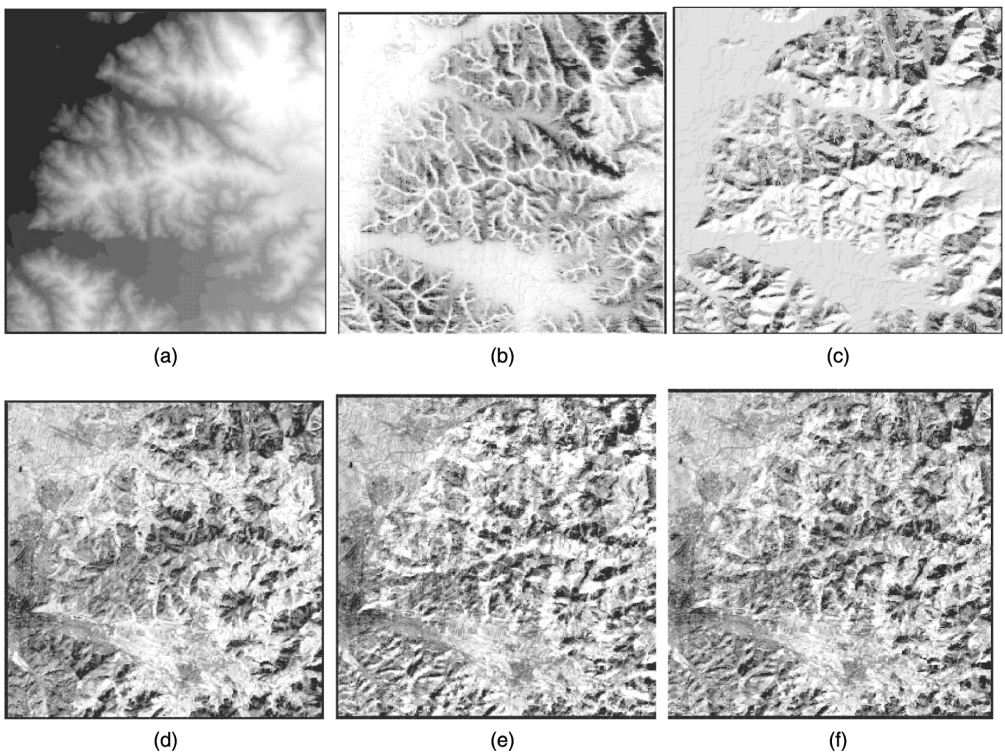


Figure 12.7. Images from Richter [21]: system to model and remove atmospheric effects and topographic effects. (a) Digital elevation model. (b) Sky view factor. (c) Illumination image. (d) Original TM band 4 image. (e) Reflectance image without processing of low-illumination areas. (f) Reflectance image with processing of low-illumination areas.

A database was compiled that contains the results of radiative transfer calculations for a wide range of weather conditions. A digital elevation model is used to obtain information about the surface elevation, slope, and orientation. Based on Lambertian assumptions, the surface reflectance in rugged terrain is calculated for the specified atmospheric conditions. Regions with extreme illumination geometries sensitive to bidirectional reflectance distribution function effects can be processed separately. This method works for high spatial resolution satellite sensor data with small swath angles. Figure 12.7 shows the results of the Richter [21] method on satellite imagery.

12.4 Heuristics and Nonphysics-Based Solutions

Heuristic and nonphysics-based approaches represent the other paradigm of atmosphere detection and correction algorithms. These algorithms do not attempt to model the physics behind the atmosphere

© 2005 by Chapman & Hall/CRC

Table 12.1. MTI bands

Band	Band #	Range (microns)	Resolution (meters)	Function
A	1	0.45–0.52	5	Blue, true color
B	2	0.52–0.60	5	Green, true color
C	3	0.62–0.68	5	Red, true color
D	4	0.76–0.86	5	Vegetation
E	5	0.86–0.89	20	Water vapor
F	6	0.91–0.97	20	Water vapor
G	7	0.99–1.04	20	Water vapor
H	8	1.36–1.39	20	Cirrus
I	9	1.54–1.75	20	Surface
O	10	2.08–2.37	20	Surface
J	11	3.49–4.10	20	Atmosphere
K	12	4.85–5.05	20	Atmosphere
L	13	8.01–8.39	20	Surface
M	14	8.42–8.83	20	Surface
N	15	10.15–10.7	20	Surface

directly. There are a wide range of algorithms, some based on empirical data, others on observations, and others that alter the sensing system itself.

Work has been done specifically to detect image areas where various atmospheric conditions are present with the idea that further processing to enhance the image in these areas could then be done. In particular, there is a body of research on cloud detection. After detection, some of these systems attempt to eliminate the clouds; most use this information for weather analysis.

Rohde et al. [22] discuss a system to detect dense clouds in daytime Multispectral Thermal Imager (MTI) Satellite images. The Rohde *et al.* [22] system uses 15 spectral bands (images). These are shown in Table 12.1 and range from visible wavelengths to longwave IR. Rhode *et al.* [22] hypothesize that clouds in the visible wavelength spectrum are bright and evenly reflect much of the wavelengths from visible to near-IR. Also, clouds appear higher in the atmosphere than most image features, and are hence colder and drier than other features. Recall that the IR spectrum is a measure of temperature. Also, for the spatial resolution of their images, clouds cover large areas of the image. Given these characteristics, they have come up with a number of parameters that can be used to classify pixels as belonging to a cloud or not.

First, the pixels are subjected to a thresholding technique in both the visible and IR ranges, based on ''clouds being bright'' and having ''even-reflection'' properties. A minimum brightness value is selected and any pixel above this brightness value in the visible range (band C) is retained as a potential cloud pixel; all others are rejected. In the IR band N, an upper limit on temperature is given, and this is used to reject pixels as potential cloud pixels.

Next, the ''whiteness'' property is tested by using a ratio of the difference of the E and C bands to their sum. Evenly reflected or ''white'' pixels will have a ratio around zero. Only pixels with ratios near zero are retained as potential cloud pixels. The last thresholding operation is done using the continuum interpolated band ratio (CIBR; see Gao and Goetz [23] for details). The CIBR can be used as a measure of the ''wetness'' (and thus ''dryness'') in the IR spectrum. The CIBR is the ratio of band F to a linear combination of bands E and G.

At this point the system has a set of pixels that conform to being ''bright,'' ''white,'' ''cold,'' and ''dry.'' What is left is to group nearby pixels and test whether they form regions large enough to indicate a cloud. This is accomplished through the removal of blobs too small to indicate clouds via a morphological opening operator (see chapter on image processing background). The final result is a binary ''cloud map'' where the nonzero pixels represent cloud pixels. The system seems to work well on the images presented, but its performance is tuned to the spatial resolution of the images and is not extensible to non-MTI-like satellite imagery. However, combinations of photometric images with IR

© 2005 by Chapman & Hall/CRC



Figure 12.8. Results of fusion system for atmosphere correction [25]. (a) One of two original foggy images, visible spectrum. (b) Fused image from two foggy images. (c) Blow-up of portion of an original foggy image. (d) Corresponding region to image (c) for fused image.

camera images could use these techniques, and a number of sensor networks with these two kinds of imaging devices do exist.

Grewe and co-workers [24,25] describe the creation of a system for correction of images in atmospheric conditions. Grewe *et al.* [25] describe a system that uses multi-image fusion to reduce atmospheric attenuation in visible spectrum images. The premise is that atmospheric conditions like fog, rain, and smoke are transient. Airflow moves the atmosphere particles in time, such that at one moment there are certain areas of a scene clearly imaged while at other moments other areas are visible. By fusing multiple images taken at different times, we may be able to improve the quality. Figure 12.8 shows typical results of the system when only two images are fused. It was discovered that the fusion engine parameters are a function of the type and level of atmosphere in the scene. Hence, this system first detects the level and type of atmospheric conditions in the image [24] and this is used to select which fusion engine to apply to the image set. The system uses a wavelet transform both to detect atmospheric conditions and to fuse multiple images of the same scene to reduce these effects. A neural network is trained to detect the level and type of atmosphere using a combination of wavelet and spatial features, like brightness, focus, and scale-changes. Typical results are shown in Figure 12.9.

Honda *et al.* [26] have developed a system to detect moving objects in a time-series sequence of images that is applied to cloud. Again, the fact that clouds move through time is taken advantage of. Here, a motion-detection algorithm is applied. This work only looks at extracting these clouds for further processing for weather applications; no correction phase takes place.

There are also systems that use sensor selection to avoid atmospheric distortions. Sweet and Tiana [2] discuss a system in which the sensors are selected to penetrate obscuring visual phenomena such as fog, snow, or smoke for the application of enabling aircraft landings in low-visibility conditions. In particular, the use of IR and MMW imaging radar is investigated. Figure 12.10 shows the components

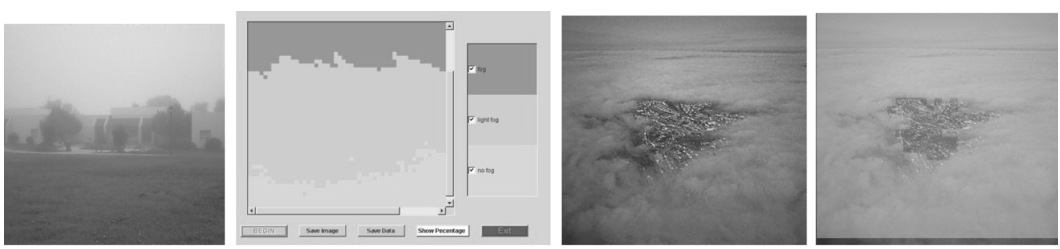


Figure 12.9. Detection of level and type of atmosphere conditions [24]. (a) Original image. (b) Application showing pseudo-colored detection image: blue = no fog, green = light fog, pink = fog. (c) Original image with cloud cover. (d) Superimposed pseudo-color detection map where green = heavy atmosphere (cloud) and gray = no atmosphere.

© 2005 by Chapman & Hall/CRC

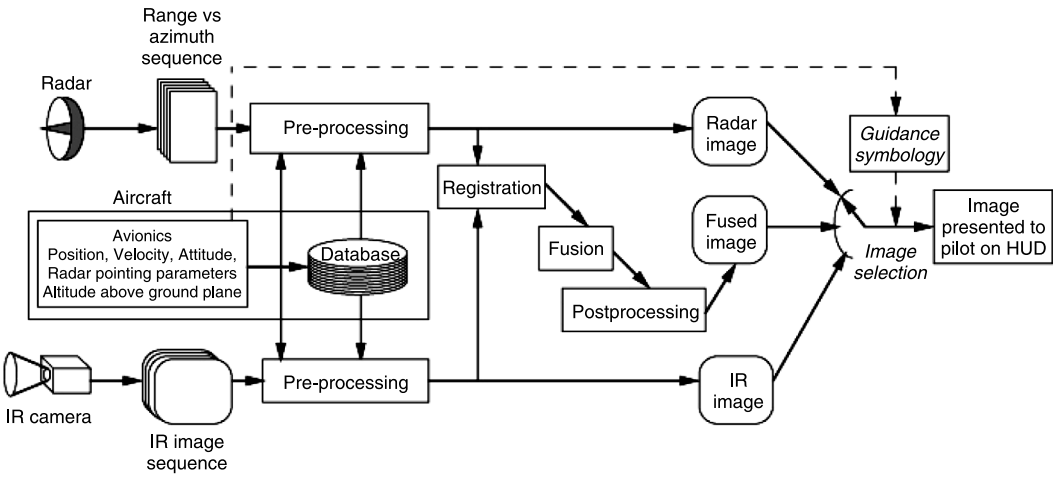


Figure 12.10. Components of a sensor system using IR and MMW radar to aid in landing aircraft in low-visibility conditions like fog [2].

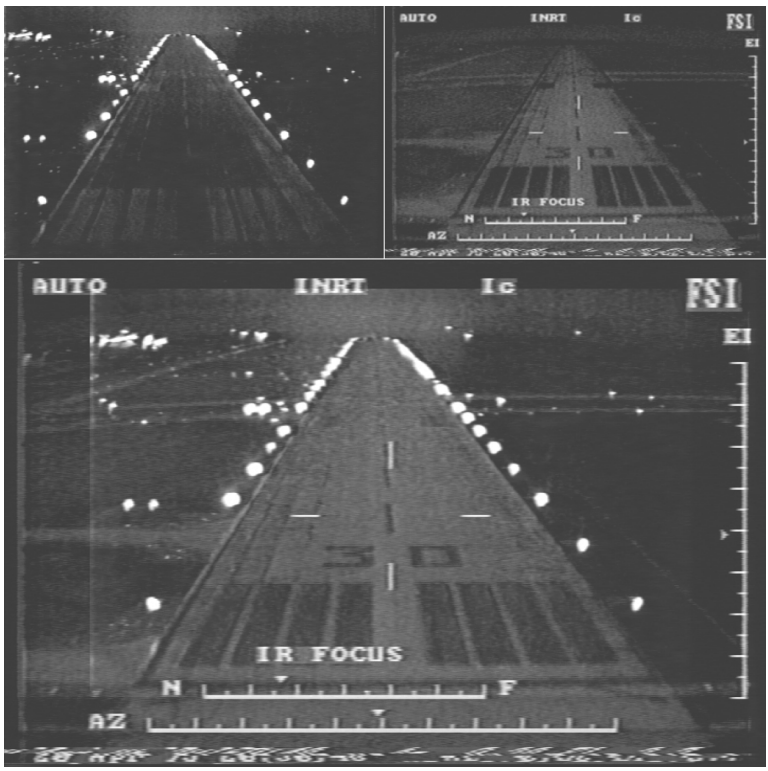


Figure 12.11. Images from Sweet and Tiana [2]: visible-spectrum image (upper left); IR image (upper right); fused IR and visible-spectrum images (lower).

of the Sweet and Tiana [2] system, and Figure 12.11 shows some images from that system. Both the IR and MMW radar images go through a preprocessing stage, followed by registration and fusion. The pilot, through a head-up display, is able to select the IR, MMW, or fused images to view for assistance in landing.

© 2005 by Chapman & Hall/CRC

Burt and Kolezynshi [27] discuss a system similar to Sweet and Tiana's [2], but MMW images are not used in the fusion process.

Another possible solution to the atmospheric correction problem is to treat it more generically as an image restoration problem. In this case, there is some similarity with blurred images and noisy images. There are a number of techniques, some discussed in the chapter on image processing, like deblurring or sharpening filters, that could improve image quality. See also Banham and Katasaggelos [28] and Kundur and Hatzinakos [29] for algorithms for deblurring images.

12.5 Conclusions

Unless you can effectively model the atmosphere, which is a very challenging problem, heuristics and nonphysics-based solutions may be the only viable solutions for a distributed sensor network. Even in the case of accurate modeling, the model would need to be dynamic and alter with the temporal changes to the atmosphere.

Very little work has been done in comparing techniques. However, Nikolakopoulos *et al.* [30] the compare two algorithms, one that explicitly models the atmosphere using a number of environmental parameters and another that uses a heuristic approach. The tests were done on multispectral data, and they found superior results for the model-based technique. Unfortunately, that study only compares two very specific algorithms, and the heuristic approach is simplistic, as it involves only histogram shifting. So, this should not be taken as an indication that model-based techniques are superior. If the assumptions and measurements made in the model are accurate, then it is reasonable to assume that model-based techniques will give good results. The information you have, the environments you wish your system to operate in, and (finally) empirical testing are what is required to select the best atmospheric correction technique for your system.

References

[1] Huxtable, B. *et al.*, A synthetic aperture radar processing system for search and rescue, *SPIE Automatic Target Recognition VII*, April 1997, 185.
[2] Sweet, B. and Tiana, C., Image processing and fusion for landing guidance, *SPIE* 2736, 84, 1996.
[3] Oakley, J. *et al.*, Enhancement of image sequences from a forward-looking airborne camera, *SPIE Image and Video Proeccing IV*, February 1996, 266.
[4] Moller, H. and Sachs, G., Synthetic vision for enhancing poor visibility flight operations, *IEEE AES Systems Magazine*, March 1994, 27.
[5] Arya, V. *et al.*, Optical fiber sensors for monitoring visibility, *SPIE Transportation Sensors and Controls: Collisions Avoidance, Traffic Management, and ITS*, November 1996, 212.
[6] Barducci, A. and Pippi, I., Retrieval of atmospheric parameters from hyperspectral image data, *International Geoscience and Remote Sensing Symposium*, July 1995, 138.
[7] Pencikowski, P., A low cost vehicle-mounted enhanced vision system comprised of a laser illuminator and range-gated camera, *SPIE Enhanced and Synthetic Vision*, April 1996, 222.
[8] Mie, G., A contribution to the optics of turbid media, especially colloidal metallic suspensions, *Annals of Physics*, 25(4), 1908, 377.
[9] Narasimhan, S. and Nayar, S., Vision and the atmosphere, *International Journal of Computer Vision*, 48(3), 2002, 233.
[10] McCartney, E., *Optics of the Atmosphere: Scattering by Molecules and Particles*, John Wiley and Sons, New York, 1975.
[11] Koschmieder, H., Theorie der horizontalen sichtweite, *Bietraege Physische Freien Atmosphere*, 12(33–53), 1924, 171–181.
[12] Curry, J. *et al.*, *Encyclopedia of Atmospheric Sciences*, Academic Press, 2002.
[13] Day, J. *et al.*, *A Field Guide to the Atmosphere*, Houghton Mifflin Co., 1998.
[14] Kyle, T., *Atmospheric Transmission, Emission and Scattering*, Pergamon Press, 1991.

© 2005 by Chapman & Hall/CRC

[15] Bohren, C., *Selected Papers on Scattering in the Atmosphere*, SPIE, 1989.
[16] Sadot, D. *et al.*, Target acquisition modeling for contrast-limited imaging: effects of atmospheric blur and image restoration, *Journal of the Optical Society of America*, 12(11), 1995, 2401.
[17] Hosgood, B., Some examples of thermal infrared applications, IPSC website, http://humanitarian-security.jrc.it/demining/infrared_files/IR_show4/sld006.htm, 2002.
[18] IPAC, NASA, What is Infrared?, http://sirtf.caltech.edu/EPO/Kidszone/infrared.html, 2003.
[19] Campbell, K., *Performance of imaging laser radar in rain and fog*, Master's Thesis, Wright-Patterson AFB, 1998.
[20] Shafer, S., Using color to separate reflection components, *Color Research and Applications*, 10, 210–218, 1985.
[21] Richter, R., Correction of satellite imagery over mountainous terrain, *Applied Optics*, 37(18), 4004–4015, 1998.
[22] Rohde, C. *et al.*, Performance of the interactive procedures for daytime detection of dense clouds in the MTI pipeline, *Algorithms for Multispectral, Hyperspectral, and Ultraspectral Imagery VII*, SPIE vol. 4381, 2001, 204.
[23] Gao, B. and Goetz A., Column atmospheric water vapor and vegation liquid water retrievals from airborne imaging spectrometer data, *Journal of Geophysical Research*, 95, 1990, 3549.
[24] Grewe, L., Detection of atmospheric conditions in images, *SPIE AeroSense: Signal Processing, Sensor Fusion and Target Recognition*, April 2001.
[25] Grewe, L. *et al.*, Atmospheric attenuation through multi-sensor fusion, *SPIE AeroSense: Sensor Fusion: Architectures, Algorithms, and Applications II*, April 1998.
[26] Honda, R. *et al.*, Mining of moving obejcts from time-series images and its application to satellite weather imagery, *Journal of Intelligent Information Systems*, 19(1), 2002, 79.
[27] Burt, P. and Kolezynski, R., Enhanced image capture through fusion, In *IEEE 4th International Conference on Computer Vision*, 1993, 173.
[28] Banham, M. and Katasaggelos, A., Spatially adaptive wavelet-based multiscale image restoration, *IEEE Transactions on Image Processing*, 5(4), 1996, 619.
[29] Kundur, D. and Hatzinakos, D., Blind image deconvolution, *IEEE Signal Processing Magazine*, 13(3), 1996, 43.
[30] Nikolakopoulos, K. *et al.*, A comparative study of different atmospheric correction algorithms over an area with complex geomorphology in western Peloponnese, Greece, *IEEE International Geoscience and Remote Sensing Symposium Proceedings*, vol. 4, 2002, 2492.

© 2005 by Chapman & Hall/CRC

13

Signal Processing and Propagation for Aeroacoustic Sensor Networks

Richard J. Kozick, Brian M. Sadler, and D. Keith Wilson

13.1 Introduction

Passive sensing of acoustic sources is attractive in many respects, including the relatively low signal bandwidth of sound waves, the loudness of most sources of interest, and the inherent difficulty of disguising or concealing emitted acoustic signals. The availability of inexpensive, low-power sensing and signal-processing hardware enables application of sophisticated real-time signal processing. Among the many applications of aeroacoustic sensors, we focus in this chapter on detection and localization of ground and air (both jet and rotary) vehicles from ground-based sensor networks. Tracking and classification are briefly considered as well.

Elaborate, aeroacoustic systems for passive vehicle detection were developed as early as World War I [1]. Despite this early start, interest in aeroacoustic sensing has generally lagged other technologies until the recent packaging of small microphones, digital signal processing, and wireless communications into compact, unattended systems. An overview of modern outdoor acoustic sensing is presented by Becker and Güdesen [2]. Experiments in the early 1990s, such as those described by Srour and Robertson [3], demonstrated the feasibility of network detection, array processing, localization, and multiple target tracking via Kalman filtering. Many of the fundmental issues and challenges described by Srour and Robertson [3] remain relevant today.

Except at very close range, the typical operating frequency range we consider is roughly 30 to 250 Hz. Below 30 Hz (the infrasonic regime) the wavelengths are greater than 10 m, so that rather large arrays may be required. Furthermore, wind noise (random pressure fluctuations induced by atmospheric turbulence) reduces the observed signal-to-noise ratio (SNR) [2]. At frequencies above several hundred hertz, molecular absorption of sound and interference between direct and ground-reflected waves attenuate received signals significantly [4]. In effect, the propagation environment acts as a low-pass filter; this is particularly evident at longer ranges.

© 2005 by Chapman & Hall/CRC

Aeroacoustics is inherently an ultra-wideband array processing problem, e.g. operating in [30, 250] Hz yields a 157% fractional bandwidth centered at 140 Hz. To process under the narrow band array assumptions will require the fractional bandwidth to be on the order of a few percent or less, limiting the bandwidth to perhaps a few hertz in this example. The wide bandwidth significantly complicates the array signal processing, including angle-of-arrival (AOA) estimation, wideband Doppler compensation, beamforming, and blind source separation (which becomes convolutional).

The typical source of interest here has a primary contribution due to rotating machinery (engines), and may include tire and/or exhaust noise, vibrating surfaces, and other contributions. Internal combustion engines typically exhibit a strong sum of harmonics acoustic signature tied to the cylinder firing rate, a feature that can be exploited in virtually all phases of signal processing. Tracked vehicles also exhibit tread slap, which can produce very strong spectral lines, while helicopters produce strong harmonic sets related to the blade rotation rates. Turbine engines, on the other hand, exhibit a much more smoothly broad spectrum and, consequently, call for different algorithmic approaches in some cases. Many heavy vehicles and aircraft are quite loud and can be detected from ranges of several kilometers or more. Ground vehicles may also produce significant seismic waves, although we do not consider multi-modal sensing or sensor fusion here.

The problem is also complicated by time-varying factors that are difficult to model, such as source signature variations resulting from acceleration/deceleration of vehicles, changing meteorological conditions, multiple soft and loud sources, aspect angle source signature dependency, Doppler shifts (with 1 Hz shifts at a 100 Hz center frequency not unusual), multipath, and so on. Fortunately, at least for many sources of interest, a piecewise stationary model is reasonable on time scales of 1 s or less, although fast-moving sources may require some form of time-varying model.

Sensor networks of interest are generally connected with wireless links, and are battery powered. Consequently, the node power budget may be dominated by the communications (radio). Therefore, a fundamental design question is how to perform distributed processing in order to reduce communication bandwidth, while achieving near optimal detection, estimation, and classification performance. We focus on this question, taking the aeroacoustic environment into account.

In particular, we consider the impact of random atmospheric inhomogeneities (primarily thermal and wind variations caused by turbulence) on the ability of an aeroacoustic sensor network to localize sources. Given that turbulence induces acoustical index-of-refraction variations several orders of magnitude greater than corresponding electromagnetic variations [5], this impact is quite significant. Turbulent scattering of sound waves causes random fluctuations in signals, as observed at a single sensor, with variations occurring on time scales from roughly one to hundreds of seconds in our frequency range of interest [6–8]. Scattering is also responsible for losses in the observed spatial coherence measured between two sensors [9–11]. The scattering may be weak or strong, which are analogous to Rician and Rayleigh fading in radio propagation, respectively.

The impact of spatial coherence loss is significant, and generally becomes worse with increasing distance between sensors. This effect, as well as practical size constraints, limits individual sensor node array apertures to perhaps a few meters. At the same time, the acoustic wavelengths λ of interest are about 1 to 10 m ($\lambda = (330\,\text{m/s})/(30\,\text{Hz}) = 11\,\text{m}$ at 30 Hz, and $\lambda = 1.32\,\text{m}$ at 250 Hz). Thus, the typical array aperture will only span a fraction of a wavelength, and accurate AOA estimation requires wideband superresolution methods. The source may generally be considered to be in the far field of these small arrays. Indeed, if it is in the near field, then the rate of change of the AOA as the source moves past the array must be considered.

The signal-coherence characteristics suggest deployment of multiple, small-baseline arrays as nodes within an overall large-baseline array (see Figure 13.7). The source is intended to be in the near field of the large-baseline array. Exploitation of this larger baseline is highly desirable, as it potentially leads to very accurate localization. We characterize this problem in terms of the atmosphere-induced spatial coherence loss, and show fundamental bounds on the ability to localize a source in such conditions. This leads to a family of localization approaches, spanning triangulation (which minimizes inter-node

© 2005 by Chapman & Hall/CRC

communication), to time-delay estimation, to fully centralized processing (which maximizes communication use and is therefore undesirable). The achievable localization accuracy depends on both the propagation conditions and the time–bandwidth product of the source.

The chapter is organized as follows. In Section 13.2 we introduce the wideband source array signal processing model, develop the atmospheric scattering model, and incorporate the scattering into the array model. We consider array signal processing in Section 13.3, including narrowband AOA estimation with scattering present. We review wideband AOA estimation techniques, and highlight various aeroacoustic wideband AOA experiments. Next, we consider localization with multiple nodes (arrays) in the presence of scattering. We develop fundamental and tight performance bounds on time delay estimation in the turbulent atmosphere, as well as bounds on localization. Localization performance is illustrated via simulation and experiments. We then briefly consider the propagation impact on detection and classification. Finally, in Section 13.4 we consider some emerging aspects and open questions.

13.2 Models for Source Signals and Propagation

In this section we present a general model for the signals received by an aeroacoustic sensor array. We begin by briefly considering models for the signals emitted by ground vehicles and aircraft in Section 13.2.1. Atmospheric phenomena affecting propagation of the signal are also summarized. In Section 13.2.2 we consider the simplest possible case for the received signals: a single nonmoving source emits a sinusoidal waveform, and the atmosphere induces no scattering (randomization of the signal). Then in Section 13.2.3 we extend the model to include the effects of scattering; in Section 13.2.4, approximate models for the scattering as a function of source range, frequency, and atmospheric conditions are presented. The model is extended to multiple sources and multiple frequencies (wideband) in Section 13.2.5.

13.2.1 Basic Considerations

As we noted in Section 13.1, the sources of interest typically have spectra that are harmonic lines, or have relatively continuous broadband spectra, or some combination. The signal processing for detection, localization, and classification is highly dependent on whether the source spectrum is harmonic or broadband. For example, broadband sources allow time-difference of arrival processing for localization, whereas harmonic sources allow differential Doppler estimation.

Various deterministic and random source models may be employed. Autoregressive (AR) processes are well suited to modeling sums of harmonics, at least for the case of a single source, and may be used for detection, Doppler estimation, filtering, AOA estimation, and so on [12–14]. Sum of harmonic models, with unknown harmonic structure, lead naturally to detection tests in the frequency domain [15].

More generally, a Gaussian random process model may be employed to describe both harmonic sets and wideband sources [16]; we adopt such a point of view here. We also assume a piecewise stationary (quasi-static) viewpoint: although the source may actually be moving, the processing interval is assumed to be short enough that the signal characteristics are nearly constant.

Four phenomena are primarily responsible for modifying the source signal to produce the signal observed at the sensor array:

1. The propagation delay from the source to the sensors.
2. Random fluctuations in the amplitude and phase of the signals caused by scattering from random inhomogeneities in the atmosphere, such as turbulence.
3. Additive noise at the sensors caused by thermal noise, wind noise, and directional interference.
4. Transmission loss caused by spreading of the wavefronts, refraction by wind and temperature gradients, ground interactions, and molecular absorption of sound energy.

© 2005 by Chapman & Hall/CRC

Thermal noise at the sensors is typically independent from sensor to sensor. In contrast, interference from an undesired source produces additive noise that is (spatially) correlated from sensor to sensor. Wind noise, which consists of low-frequency turbulent pressure fluctuations intrinsic to the atmospheric flow (and, to a lesser extent, flow distortions induced by the microphone itself [2,17]), exhibits high spatial correlation over distances of several meters [18].

The transmission loss (TL) is defined as the diminishment in sound energy from a reference value S_{ref}, which would hypothetically be observed in free space at 1 m from the source, to the actual value observed at the sensor S. To a first approximation, the sound energy spreads spherically; that is, it diminishes as the inverse of the squared distance from the source. In actuality the TL for a sound wave propagating near the ground involves many complex, interacting phenomena, so that the spherical spreading condition is rarely observed in practice, except perhaps within the first 10 to 30 m [4]. Fortunately, several well-refined and accurate numerical procedures for calculating TL have been developed [19]. For simplicity, here we model S as a deterministic parameter, which is reasonable when the state of the atmosphere does not change dramatically during the data collection.

Particularly significant to the present discussion is the second phenomenon in the above list, namely scattering by turbulence. The turbulence consists of random atmospheric motions occurring on time scales from seconds to several minutes. Scattering from these motions causes random fluctuations in the complex signals at the individual sensors and diminishes the cross-coherence of signals between sensors. The effects of scattering on array performance will be analyzed in Section 13.3.

The sinusoidal source signal that is measured at the reference distance of 1 m from the source is written

$$S_{\text{ref}}(t) = \sqrt{S_{\text{ref}}}\,\cos(2\pi f_{\text{o}} t + \chi) \tag{13.1}$$

where the frequency of the tone is $f_{\text{o}} = \omega_{\text{o}}/(2\pi)$ Hz, the period is T_{o} s, the phase is χ, and the amplitude is $\sqrt{S_{\text{ref}}}$. The sound waves propagate with wavelength $\lambda = c/f_{\text{o}}$, where c is the speed of sound. The wavenumber is $k = 2\pi/\lambda = \omega_{\text{o}}/c$. We will represent sinusoidal and narrowband signals by their complex envelope, which may be defined in two ways, as in (13.2):

$$\mathcal{C}\{s_{\text{ref}}(t)\} = \widetilde{s}_{\text{ref}}(t) = s_{\text{ref}}^{(\text{I})}(t) + \text{j}s_{\text{ref}}^{(\text{Q})}(t) = \left[s_{\text{ref}}(t) + \text{j}\mathcal{H}\{s_{\text{ref}}(t)\}\right]\exp(-\text{j}2\pi f_{\text{o}} t) \tag{13.2}$$

$$= \sqrt{S_{\text{ref}}}\,\exp(\text{j}\chi) \tag{13.3}$$

We will represent the complex envelope of a quantity with the notation $\mathcal{C}\{\cdot\}$ or $\widetilde{(\cdot)}$, the in-phase component with $(\cdot)^{(\text{I})}$, the quadrature component with $(\cdot)^{(\text{Q})}$, and the Hilbert transform with $\mathcal{H}\{\cdot\}$. The in-phase (I) and quadrature (Q) components of a signal are obtained by the processing in Figure 13.2. The fast fourier transform (FFT) is often used to approximate the processing in Figure 13.2 for a finite block of data, where the real and imaginary parts of the FFT coefficient at frequency f_{o} are proportional to the I and Q components respectively. The complex envelope of the sinusoid in (13.1) is given by (13.3), which is not time-varying, so the average power is $\left|\widetilde{s}_{\text{ref}}(t)\right|^2 = S_{\text{ref}}$.

It is easy to see for the sinusoidal signal in Equation (13.1) that shifting $s_{\text{ref}}(t)$ in time causes a phase shift in the corresponding complex envelope, i.e. $\mathcal{C}\{s_{\text{ref}}(t-\tau_{\text{o}})\} = \exp(-\text{j}2\pi f_{\text{o}}\tau_{\text{o}})\widetilde{s}_{\text{ref}}(t)$. A similar property is true for *narrowband* signals whose frequency spectrum is confined to a bandwidth B Hz around a center frequency f_{o} Hz, where $B \ll f_{\text{o}}$. For a narrowband signal $z(t)$ with complex envelope $\widetilde{z}(t)$, a shift in time is well approximated by a phase shift in the corresponding complex envelope:

$$\mathcal{C}\{z(t-\tau_{\text{o}})\} \approx \exp(-\text{j}2\pi f_{\text{o}}\tau_{\text{o}})\widetilde{z}(t) \quad \text{(narrowband approximation)} \tag{13.4}$$

© 2005 by Chapman & Hall/CRC

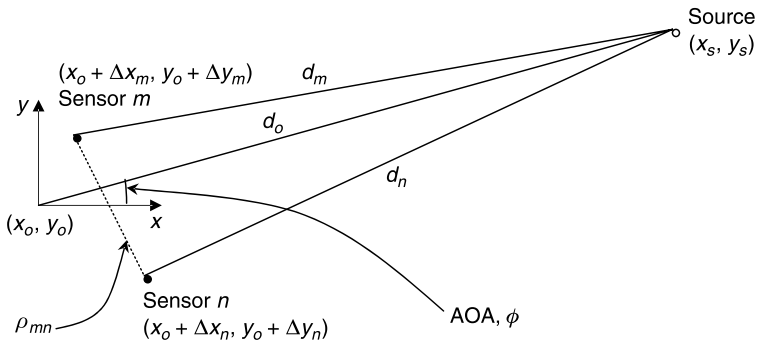


Figure 13.1. Geometry of source and sensor locations.

Equation (13.4) is the well-known Fourier transform relationship between shifts in time and phase shifts that are linearly proportional to frequency. The approximation is accurate when the frequency band is narrow enough so that the linearly increasing phase shift is close to $\exp(-j2\pi f_o \tau_o)$ over the band.

The source and array geometry is illustrated in Figure 13.1. The source is located at coordinates (x_s, y_s) in the (x, y) plane. The array contains N sensors, with sensor n located at $(x_o + \Delta x_n, y_o + \Delta y_n)$, where (x_o, y_o) is the center of the array and $(\Delta x_n, \Delta y_n)$ is the relative sensor location. The propagation time from the source to the array center is

$$\tau_o = \frac{d_o}{c} = \frac{1}{c}\left[(x_s - x_o)^2 + (y_s - y_o)^2\right]^{1/2} \tag{13.5}$$

where d_o is the distance from the source to the array center. The propagation time from the source to sensor n is

$$\tau_n = \frac{d_n}{c} = \frac{1}{c}\left[(x_s - x_o - \Delta x_n)^2 + (y_s - y_o - \Delta y_n)^2\right]^{1/2} \tag{13.6}$$

Let us denote the array diameter by $L = \max\{\rho_{mn}\}$, where ρ_{mn} is the separation between sensors m and n, as shown in Figure 13.1. The source is in the *far field* of the array when the source distance satisfies $d_o \gg L^2/\lambda$, in which case Equation (13.6) may be approximated with the first term in the Taylor series $(1 + u)^{1/2} \approx 1 + u/2$. Then $\tau_n \approx \tau_o + \tau_{o,n}$ with error that is much smaller than the source period T_o, where

$$\tau_{o,n} = -\frac{1}{c}\left(\frac{x_s - x_o}{d_o}\Delta x_n + \frac{y_s - y_o}{d_o}\Delta y_n\right) = -\frac{1}{c}\left[(\cos\ \phi)\Delta x_n + (\sin\ \phi)\Delta y_n\right] \tag{13.7}$$

The angle ϕ is the azimuth bearing, or AOA, as shown in Figure 13.1. In the far field, the spherical wavefront is approximated as a plane wave over the array aperture, so the bearing ϕ contains the available information about the source location. For array diameters $L < 2$ m and tone frequencies $f_o < 200$ Hz so that $\lambda > 1.5$ m, the quantity $L^2/\lambda < 2.7$ m. Thus the far field is valid for source distances on the order of tens of meters. For smaller source distances and/or larger array apertures, the curvature of the wavefront over the array aperture must be included in τ_n according to Equation (13.6). We develop the model for the far-field case in the next section. However, the extension to the near field is

© 2005 by Chapman & Hall/CRC

easily accomplished by redefining the array response vector ($\mathbf{a}$ in Equation (13.20)) to include the wavefront curvature with $a_n = \exp(-\mathrm{j}2\pi f_o \tau_n)$.

13.2.2 Narrowband Model with No Scattering

Here, we present the model for the signals impinging on the sensor array when there is no scattering. Using the far-field approximation, the noisy measurements at the sensors are

$$z_n(t) = s_n\,(t - \tau_o - \tau_{o,n}) + w_n(t), \quad n = 1, \ldots, N \tag{13.8}$$

In the absence of scattering, the signal components are pure sinusoids:

$$s_n(t) = \sqrt{S}\,\cos\bigl(2\pi f_o t + \chi\bigr) \tag{13.9}$$

The $w_n(t)$ are additive, white, Gaussian noise (AWGN) processes that are real-valued, continuous-time, zero-mean, jointly wide-sense stationary, and mutually uncorrelated at distinct sensors with power spectral density (PSD) $(\mathcal{N}_o/2)$ W/Hz. That is, the noise correlation properties are

$$E\{w_n(t)\} = 0, \quad -\infty < t < \infty \quad n = 1, \ldots, N \tag{13.10}$$

$$r_{w,mn}(\xi) = E\{w_m(t+\xi)w_n(t)\} = r_w(\xi)\,\delta_{mn} \tag{13.11}$$

where $E\{\cdot\}$ denotes expectation and $r_w(\xi) = (\mathcal{N}_o/2)\,\delta(\xi)$ is the noise autocorrelation function that is common at all sensors. The Dirac delta function is $\delta(\cdot)$, and the Kronecker delta function is $\delta_{mn} = 1$ if $m = n$ and 0 otherwise. As noted above, modeling the noise as spatially white may be inaccurate if wind noise or interfering sources are present in the environment. The noise PSD is

$$G_w(f) = \mathcal{F}\{r_w(\xi)\} = \frac{\mathcal{N}_o}{2} \tag{13.12}$$

where $\mathcal{F}\{\cdot\}$ denotes Fourier transform. With no scattering, the complex envelope of $z_n(t)$ in Equations (13.8) and (13.9) is, using Equation (13.4)

$$\begin{aligned}\widetilde{z}_n(t) &= \exp\bigl[-\mathrm{j}(\omega_o\tau_o + \omega_o\tau_o,\, n)\bigr]\widetilde{s}_n(t) + \widetilde{w}_n(t) \\ &= \sqrt{S}\,\exp\bigl[\mathrm{j}(\chi - \omega_o\tau_o)\bigr]\,\exp\,[-\mathrm{j}\,\omega_o\tau_{o,n}] + \widetilde{w}_n(t)\end{aligned} \tag{13.13}$$

where the complex envelope of the narrowband source component is

$$\widetilde{s}_n(t) = \sqrt{S}\,\mathrm{e}^{\mathrm{j}\chi}, \quad n = 1, \ldots, N \quad \text{(no scattering)} \tag{13.14}$$

We assume that the complex envelope is low-pass filtered with bandwidth from $[-B/2,\, B/2]$ Hz, e.g. as in Figure 13.2. Assuming that the low-pass filter is ideal, the complex envelope of the noise, $\widetilde{w}_n(t)$, has PSD and correlation

$$G_{\widetilde{w}}(f) = (2\mathcal{N}_o)\,\mathrm{rect}\left(\frac{f}{B}\right) \tag{13.15}$$

© 2005 by Chapman & Hall/CRC

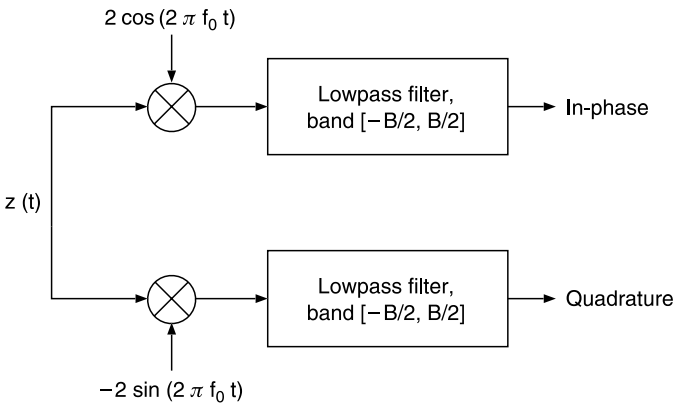


Figure 13.2. Processing to obtain in-phase and quadrature components, $z^{(\mathrm{I})}(t)$ and $z^{(\mathrm{Q})}(t)$.

$$r_{\tilde{w}}(\xi) = E\{\widetilde{w}_n(t+\xi)\widetilde{w}_n(t)^*\} = \mathcal{F}^{-1}\{G_{\tilde{w}}(f)\} = (2\mathcal{N}_o B)\,\mathrm{sinc}\,(B\xi) \tag{13.16}$$

$$r_{\tilde{w},mn}(\xi) = E\{\widetilde{w}_m(t+\xi)\widetilde{w}_n(t)^*\} = r_{\tilde{w}}(\xi)\,\delta_{mn} \tag{13.17}$$

where $(\cdot)^*$ denotes complex conjugate, $\mathrm{rect}(u) = 1$ for $-1/2 < u < 1/2$ and 0 otherwise, and $\mathrm{sinc}(u) = \sin(\pi u)/(\pi u)$. Note that the noise samples are uncorrelated (and independent since Gaussian) at sample times spaced by $1/B$ s. In practice, the noise PSD $G_{\tilde{w}}(f)$ is neither flat nor perfectly band-limited as in Equation (13.15). However, the low-pass filtering to bandwidth B Hz implies that the noise samples have decreasing correlation for time spacing greater than $1/B$ s.

Let us define the vectors

$$\widetilde{\mathbf{z}}(t) = \begin{bmatrix} \widetilde{z}_1(t) \\ \vdots \\ \widetilde{z}_N(t) \end{bmatrix}, \quad \widetilde{\mathbf{s}}(t) = \begin{bmatrix} \widetilde{s}_1(t) \\ \vdots \\ \widetilde{s}_N(t) \end{bmatrix}, \quad \widetilde{\mathbf{w}}(t) = \begin{bmatrix} \widetilde{w}_1(t) \\ \vdots \\ \widetilde{w}_N(t) \end{bmatrix} \tag{13.18}$$

Then, using (13.13) with (13.7):

$$\widetilde{\mathbf{z}}(t) = \sqrt{S}\,\exp\big[\,\mathrm{j}(\chi - \omega_o\tau_o)\big]\mathbf{a} + \widetilde{\mathbf{w}}(t) = \sqrt{S}\,\mathrm{e}^{\mathrm{j}\theta}\,\mathbf{a} + \widetilde{\mathbf{w}}(t) \tag{13.19}$$

where $\mathbf{a}$ is the array steering vector (or array manifold)

$$\mathbf{a} = \begin{bmatrix} \exp\big[\,jk\big((\cos\phi)\Delta x_1 + (\sin\phi)\Delta y_1\big)\big] \\ \vdots \\ \exp\big[\,jk\big((\cos\phi)\Delta x_N + (\sin\phi)\Delta y_N\big)\big] \end{bmatrix} \tag{13.20}$$

with $k = \omega_o/c$. Note that the steering vector $\mathbf{a}$ depends on the frequency ω_o, the sensor locations $(\Delta x_n, \Delta y_n)$, and the source bearing ϕ. The common phase factor at all of the sensors, $\exp\big[\mathrm{j}(\chi - \omega_o\tau_o)\big] = \exp\big[\mathrm{j}(\chi - kd_o)\big]$, depends on the phase of the signal emitted by the source (χ) and the propagation distance to the center of the array (kd_o). We simplify the notation and define

$$\theta \triangleq \chi - kd_o \tag{13.21}$$

which is a deterministic parameter.

© 2005 by Chapman & Hall/CRC

In preparation for the introduction of scattering into the model, let us write expressions for the first- and second-order moments of the vectors $\widetilde{\mathbf{s}}(t)$ and $\widetilde{\mathbf{z}}(t)$. Let $\mathbf{1}$ be an $N \times 1$ vector of ones, $\mathbf{R}_{\widetilde{\mathbf{z}}}(\xi) = E\{\widetilde{\mathbf{z}}(t+\xi)\widetilde{\mathbf{z}}(t)^{\dagger}\}$ be the $N \times N$ cross-correlation function matrix with (m, n) element $r_{\widetilde{z},mn}(\xi) = E\{\widetilde{z}_m(t+\xi)\widetilde{z}_n(t)^*\}$, and $\mathbf{G}_{\widetilde{\mathbf{z}}}(f) = \mathcal{F}\left\{\mathbf{R}_{\widetilde{\mathbf{z}}}(\xi)\right\}$ be the cross-spectral density (CSD) matrix; then

$$E\{\widetilde{\mathbf{s}}(t)\} = \sqrt{S}\mathrm{e}^{\mathrm{j}\chi}\mathbf{1} \qquad E\{\widetilde{\mathbf{z}}(t)\} = \sqrt{S}\mathrm{e}^{\mathrm{j}\theta}\mathbf{a} \tag{13.22}$$

$$\mathbf{R}_{\widetilde{\mathbf{s}}}(\xi) = S\mathbf{1}\mathbf{1}^{\mathrm{T}} \qquad \mathbf{R}_{\widetilde{\mathbf{z}}}(\xi) = S\,\mathbf{a}\mathbf{a}^{\dagger} + r_{\widetilde{w}}(\xi)\mathbf{I} \tag{13.23}$$

$$\mathbf{G}_{\widetilde{\mathbf{s}}}(f) = S\mathbf{1}\mathbf{1}^{\mathrm{T}}\delta(f) \qquad \mathbf{G}_{\widetilde{\mathbf{z}}}(f) = S\mathbf{a}\mathbf{a}^{\dagger}\delta(f) + G_{\widetilde{w}}(f)\mathbf{I} \tag{13.24}$$

$$E\{\widetilde{\mathbf{s}}(t)\widetilde{\mathbf{s}}(t)^{\dagger}\} = \mathbf{R}_{\widetilde{\mathbf{s}}}(0) = S\mathbf{1}\mathbf{1}^{T} \quad E\{\widetilde{\mathbf{z}}(t)\widetilde{\mathbf{z}}(t)^{\dagger}\} = \mathbf{R}_{\widetilde{\mathbf{z}}}(0) = S\,\mathbf{a}\mathbf{a}^{\dagger} + \sigma_{\widetilde{w}}^2\mathbf{I} \tag{13.25}$$

where $(\cdot)^{\mathrm{T}}$ denotes transpose, $(\cdot)^*$ denotes complex conjugate, $(\cdot)^{\dagger}$ denotes complex conjugate transpose, $\mathbf{I}$ is the $N \times N$ identity matrix, and $\sigma_{\widetilde{w}}^2$ is the variance of the noise samples:

$$\sigma_{\widetilde{w}}^2 = E\left\{\left|\widetilde{\mathbf{w}}(t)\right|^2\right\} = r_{\widetilde{w}}(0) = 2\mathcal{N}_{\mathrm{o}}B \tag{13.26}$$

Note from Equation (13.24) that the PSD at each sensor contains a spectral line, since the source signal is sinusoidal. Note from Equation (13.25) that, at each sensor, the average power of the signal component is S, so the SNR at each sensor is

$$\mathrm{SNR} = \frac{S}{\sigma_{\widetilde{w}}^2} = \frac{S}{2\mathcal{N}_{\mathrm{o}}B} \tag{13.27}$$

The complex envelope vector $\widetilde{\mathbf{z}}(t)$ is typically sampled at a rate $f_{\mathrm{s}} = B$ samples/s, so the samples are spaced by $T_{\mathrm{s}} = 1/f_{\mathrm{s}} = 1/B$ s:

$$\widetilde{\mathbf{z}}(iT_{\mathrm{s}}) = \sqrt{S}\,\mathrm{e}^{\mathrm{j}\theta}\,\mathbf{a} + \widetilde{\mathbf{w}}(iT_{\mathrm{s}}), \quad i = 0, \ldots, T-1 \tag{13.28}$$

According to Equation (13.17), the noise samples are spatially independent as well as temporally independent, since $r_{\widetilde{w}}(iT_{\mathrm{s}}) = r_{\widetilde{w}}(i/B) = 0$. Thus the vectors $\widetilde{\mathbf{z}}(0)$, $\widetilde{\mathbf{z}}(T_{\mathrm{s}})$, $\ldots$, $\widetilde{\mathbf{z}}((T-1)T_{\mathrm{s}})$ in Equation (13.28) are independent and identically distributed (iid) with complex normal distribution, which we denote by $\widetilde{\mathbf{z}}(iT_{\mathrm{s}}) \sim \mathrm{CN}(\mathbf{m}_{\widetilde{\mathbf{z}}}, \mathbf{C}_{\widetilde{\mathbf{z}}})$, with mean and covariance matrix

$$\mathbf{m}_{\widetilde{\mathbf{z}}} = \sqrt{S}\,\mathrm{e}^{\mathrm{j}\theta}\,\mathbf{a} \quad \text{and} \quad \mathbf{C}_{\widetilde{\mathbf{z}}} = \sigma_{\widetilde{w}}^2\,\mathbf{I} \quad \text{(no scattering)} \tag{13.29}$$

The joint probability density function for $\mathrm{CN}(\mathbf{m}_{\widetilde{\mathbf{z}}}, \mathbf{C}_{\widetilde{\mathbf{z}}})$ is given by [20]

$$f(\widetilde{\mathbf{z}}) = \frac{1}{\pi^N \det(\mathbf{C}_{\widetilde{\mathbf{z}}})} \exp\left[-(\widetilde{\mathbf{z}} - \mathbf{m}_{\widetilde{\mathbf{z}}})^{\dagger}\,\mathbf{C}_{\widetilde{\mathbf{z}}}^{-1}(\widetilde{\mathbf{z}} - \mathbf{m}_{\widetilde{\mathbf{z}}})\right] \tag{13.30}$$

where "det" denotes determinant. In the absence of scattering, the information about the source location (bearing) is contained in the mean of the sensor observations. If the T time samples in Equation (13.28) are coherently averaged, then the resulting SNR per sensor is T times that in Equation (13.27),

© 2005 by Chapman & Hall/CRC

so $\text{SNR}' = T(S/\sigma_{\tilde{w}}^2) = T[S/(2\mathcal{N}_o/T_s)] = \mathcal{T}S/(2\mathcal{N}_o)$, where $\mathcal{T} = T\,T_s$ is the total observation time, in seconds.

13.2.3 Narrowband Model with Scattering

Next, we include the effects of scattering by atmospheric turbulence in the model for the signals measured at the sensors in the array. As mentioned earlier, the scattering introduces random fluctuations in the signals and diminishes the cross-coherence between the array elements. The formulation we present for the scattering effects was developed by Wilson, Collier and coworkers [11,21–26]. The reader may refer to these studies for details about the physical modeling and references to additional primary source material. Several assumptions and simplifications are involved in the formulation: (1) the propagation is line-of-sight (no multipath), (2) the additive noise is independent from sensor to sensor, and (3) the random fluctuations caused by scattering are complex, circular, Gaussian random processes with partial correlation between the sensors.

The line-of-sight propagation assumption is consistent with Section 13.2.2 and is reasonable for propagation over fairly flat, open terrain in the frequency range of interest here (below several hundred hertz). A significant acoustic multipath may result from reflections off hard objects, such as buildings, trees, and (sometimes) the ground. A multipath can also result from refraction of sound waves by vertical gradients in the wind and temperature.

By assuming independent, additive noise, we ignore the potential spatial correlation of wind noise and interference from other undesired sources. This restriction may be averted by extending the models to include spatially correlated additive noise, although the signal processing may be more complicated in this case.

Modeling of the scattered signals as complex, circular, Gaussian random processes is a substantial improvement on the constant signal model (Section 13.2.2), but it is, nonetheless, rather idealized. Waves that have propagated through a random medium can exhibit a variety of statistical behaviors, depending on such factors as the strength of the turbulence, the propagation distance, and the ratio of the wavelength to the predominant eddy size [5,27]. Experimental studies [8,28,29] conducted over short horizontal propagation distances with frequencies below 1000 Hz demonstrate that the effect of turbulence is highly significant, with phase variations much larger than 2π radians and deep fades in amplitude often developing. The measurements demonstrate that the Gaussian model is valid in many conditions, although non-Gaussian scattering characterized by large phase but small amplitude variations is observed at some frequencies and propagation distances. The Gaussian model applies in many cases of interest, and we apply it in this chapter. The effect of non-Gaussian signal scattering on aeroacoustic array performance remains to be determined.

The scattering modifies the complex envelope of the signals at the array by spreading a portion of the power from the (deterministic) mean component into a zero-mean random process with a PSD centered at 0 Hz. We assume that the bandwidth of the scattered signal, which we denote by B_ν, is much smaller than the tone frequency f_o. The *saturation* parameter [25,26], denoted by $\Omega \in [0,\ 1]$, defines the fraction of average signal power that is scattered from the mean into the random component. The scattering may be *weak* ($\Omega \approx 0$) or *strong* ($\Omega \approx 1$), which are analogous to Rician and Rayleigh fading respectively in the radio propagation literature. The modification of Equations (13.8), (13.9), (13.13), and (13.14) to include scattering is as follows, where $\widetilde{z}_n(t)$ is the signal measured at sensor n:

$$\widetilde{z}_n(t) = \exp\left[-\mathrm{j}\left(\omega_o\tau_o + \omega_o\tau_{o,n}\right)\right]\widetilde{s}_n(t) + \widetilde{w}_n(t) \tag{13.31}$$

$$\widetilde{s}_n(t) = \sqrt{(1-\Omega)S}\,\mathrm{e}^{\mathrm{j}\chi} + \widetilde{v}_n(t)\,\mathrm{e}^{\mathrm{j}\chi}, \quad n = 1, \ldots, N \text{ (with scattering)} \tag{13.32}$$

In order to satisfy conservation of energy with $E\{|\widetilde{s}_n(t)|^2\} = S$, the average power of the scattered component must be $E\{|\widetilde{v}_n(t)|^2\} = \Omega\,S$. The value of the saturation Ω and the correlation properties of the vector of scattered processes, $\widetilde{\mathbf{v}}(t) = [\widetilde{v}_1(t), \ldots, \widetilde{v}_N(t)]^{\mathrm{T}}$, depend on the source distance d_o and the

© 2005 by Chapman & Hall/CRC

meteorological conditions. The vector of scattered processes $\widetilde{\mathbf{v}}(t)$ and the additive noise vector $\widetilde{\mathbf{w}}(t)$ contain zero-mean, jointly wide-sense stationary, complex, circular Gaussian random processes. The scattered processes and the noise are modeled as independent, $E\{\widetilde{\mathbf{v}}(t+\xi)\widetilde{\mathbf{w}}(t)^{\dagger}\} = \mathbf{0}$. The noise is described by Equations (13.15)–(13.17), while the saturation Ω and statistics of $\widetilde{\mathbf{v}}(t)$ are determined by the "extinction coefficients" of the first and second moments of $\widetilde{\mathbf{s}}(t)$. As will be discussed in Section 13.2.4, approximate analytical models for the extinction coefficients are available from physical modeling of the turbulence in the atmosphere. In the remainder of this section we define the extinction coefficients and relate them to Ω and the statistics of $\widetilde{\mathbf{v}}(t)$, thereby providing models for the sensor array data that include turbulent scattering by the atmosphere.

We denote the extinction coefficients for the first and second moments of $\widetilde{\mathbf{s}}(t)$ by μ and $\nu(\rho_{mn})$ respectively, where ρ_{mn} is the distance between sensors m and n (see Figure 13.1). The extinction coefficients are implicitly defined as follows:

$$E\{\widetilde{s}_n(t)\} = \sqrt{(1-\Omega)S}\,\mathrm{e}^{\mathrm{j}\chi} \triangleq \sqrt{S}\,\mathrm{e}^{\mathrm{j}\chi}\,\mathrm{e}^{-\mu d_{\mathrm{o}}} \tag{13.33}$$

$$r_{\tilde{s},mn}(0) = E\{\widetilde{s}_m(t)\widetilde{s}_n(t)^*\} = (1-\Omega)S + r_{\tilde{v},mn}(0) \triangleq S\mathrm{e}^{-\nu(\rho_{mn})d_{\mathrm{o}}} \tag{13.34}$$

where

$$r_{\tilde{s},mn}(\xi) = E\{\widetilde{s}_m(t+\xi)\widetilde{s}_n(t)^*\} = (1-\Omega)S + r_{\tilde{v},mn}(\xi) \tag{13.35}$$

The right sides of Equations (13.33) and (13.34) are the first and second moments *without* scattering, from Equations (13.22) and (13.23), respectively, multiplied by a factor that decays exponentially with increasing distance d_{o} from the source. From Equation (13.33), we obtain

$$\sqrt{(1-\Omega)} = \mathrm{e}^{-\mu d_{\mathrm{o}}} \quad \text{and} \quad \Omega = 1 - \mathrm{e}^{-2\mu d_{\mathrm{o}}} \tag{13.36}$$

Also, by conservation of energy with $m = n$ in Equation (13.34), adding the average powers in the unscattered and scattered components of $\widetilde{s}_n(t)$ must equal S, so

$$r_{\tilde{s}}(0) = E\left\{\left|\widetilde{s}_n(t)\right|^2\right\} = \mathrm{e}^{-2\mu d_{\mathrm{o}}}S + r_{\tilde{v}}(0) = S \tag{13.37}$$

$$\Longrightarrow r_{\tilde{v}}(0) = E\left\{\left|\widetilde{v}_n(t)\right|^2\right\} = \int_{-\infty}^{\infty} G_{\tilde{v}}(f)\,\mathrm{d}f = \left(1 - \mathrm{e}^{-2\mu \mathrm{d}_{\mathrm{o}}}\right) S = \Omega S \tag{13.38}$$

where $r_{\tilde{v}}(\xi) = E\{\widetilde{v}_n(t+\xi)\widetilde{v}_n(t)^*\}$ is the autocorrelation function (which is the same for all n) and $G_{\tilde{v}}(f)$ is the corresponding PSD. Therefore, for source distances $d_{\mathrm{o}} \ll 1/(2\mu)$, the saturation $\Omega \approx 0$ and most of the energy from the source arrives at the sensor in the unscattered (deterministic mean) component of $\widetilde{s}_n(t)$. For source distances $d_{\mathrm{o}} \gg 1/(2\mu)$, the saturation $\Omega \approx 1$ and most of the energy arrives in the scattered (random) component.

Next, we use Equation (13.34) to relate the correlation of the scattered signals at sensors m and n, $r_{\tilde{v},mn}(\xi)$, to the second moment extinction coefficient $\nu(\rho_{mn})$. Since the autocorrelation of $\widetilde{v}_n(t)$ is identical at each sensor n and equal to $r_{\tilde{v}}(\xi)$, and assuming that the PSD $G_{\tilde{v}}(f)$ occupies a narrow bandwidth centered at 0 Hz, the cross-correlation and cross-spectral density satisfy

$$r_{\tilde{v},mn}(\xi) = \gamma_{mn}\, r_{\tilde{v}}(\xi) \quad \text{and} \quad G_{\tilde{v},mn}(f) = \mathcal{F}\{r_{\tilde{v},mn}(\xi)\} = \gamma_{mn} G_{\tilde{v}}(f) \tag{13.39}$$

where $|\gamma_{mn}| \leq 1$ is a measure of the coherence between $\widetilde{v}_m(t)$ and $\widetilde{v}_n(t)$. The definition of γ_{mn} as a *constant* includes an approximation that the coherence does not vary with frequency, which is

© 2005 by Chapman & Hall/CRC

reasonable when the bandwidth of $G_{\tilde{v}}(f)$ is narrow. Although systematic studies of the coherence time of narrowband acoustic signals have not been made, data and theoretical considerations (such as in [27, Sec. 8.4]) are consistent with values ranging from tens of seconds to several minutes in the frequency range [50, 250] Hz. Therefore, the bandwidth of $G_{\tilde{v}}(f)$ may be expected to be less than 1 Hz. The bandwidth B in the low-pass filters for the complex amplitude in Figure 13.2 should be chosen to be greater than or equal to the bandwidth of $G_{\tilde{v}}(f)$. We assume that γ_{mn} in Equation (13.39) is real-valued and nonnegative, which implies that phase fluctuations at sensor pairs are not biased toward positive or negative values. Then, using Equation (13.39) with Equations (13.38) and (13.36) in Equation (13.34) yields the following relation between γ_{mn} and μ, ν:

$$\gamma_{mn} = \frac{e^{-\nu(\rho_{mn})d_o} - e^{-2\mu d_o}}{1 - e^{-2\mu d_o}}, \quad m, n = 1, \ldots, N \tag{13.40}$$

We define $\mathbf{\Gamma}$ as the $N \times N$ matrix with elements γ_{mn}. The second moment extinction coefficient $\nu(\rho_{mn})$ is a monotonically increasing function, with $\nu(0) = 0$ and $\nu(\infty) = 2\mu$, so $\gamma_{mn} \in [0, 1]$.

Combining Equations (13.31) and (13.32) into vectors, and using Equation (13.36) yields

$$\tilde{\mathbf{z}}(t) = \sqrt{S}\, e^{j\theta}\, e^{-\mu d_o}\, \mathbf{a} + e^{j\theta}\, \mathbf{a} \circ \tilde{\mathbf{v}}(t) + \tilde{\mathbf{w}}(t) \tag{13.41}$$

where θ is defined in Equation (13.21) and $\mathbf{a}$ is the array steering vector in Equation (13.20), where $\circ$ denotes element-wise product between matrices. We define the matrix $\mathbf{B}$ with elements

$$B_{mn} = \exp[-\nu(\rho_{mn})\, d_o] \tag{13.42}$$

and then we can extend the second-order moments in Equations (13.22)–(13.25) to the case with scattering as

$$E\{\tilde{\mathbf{z}}(t)\} = e^{-\mu d_o} \sqrt{S} e^{j\theta}\, \mathbf{a} \triangleq \mathbf{m}_{\tilde{\mathbf{z}}} \tag{13.43}$$

$$\mathbf{R}_{\tilde{\mathbf{z}}}(\xi) = e^{-2\mu d_o} S\, \mathbf{a}\mathbf{a}^\dagger + S\left[\mathbf{B} \circ \left(\mathbf{a}\mathbf{a}^\dagger\right) - e^{-2\mu d_o} \mathbf{a}\mathbf{a}^\dagger\right] \frac{r_{\tilde{v}}(\xi)}{S(1 - e^{-2\mu d_o})} + r_{\tilde{w}}(\xi)\mathbf{I} \tag{13.44}$$

$$\mathbf{G}_{\tilde{\mathbf{z}}}(f) = e^{-2\mu d_0}\, S\, \mathbf{a}\mathbf{a}^\dagger\, \delta(f) + S\left[\mathbf{B} \circ \left(\mathbf{a}\mathbf{a}^\dagger\right) - e^{-2\mu d_o} \mathbf{a}\mathbf{a}^\dagger\right] \frac{G_{\tilde{v}}(f)}{S(1 - e^{-2\mu d_o})} + G_{\tilde{w}}(f)\mathbf{I} \tag{13.45}$$

$$E\{\tilde{\mathbf{z}}(t)\tilde{\mathbf{z}}(t)^\dagger\} = \mathbf{R}_{\tilde{\mathbf{z}}}(0) = S\mathbf{B} \circ \left(\mathbf{a}\mathbf{a}^\dagger\right) + \sigma_{\tilde{w}}^2 \mathbf{I} = \mathbf{C}_{\tilde{\mathbf{z}}} + \mathbf{m}_{\tilde{\mathbf{z}}} \mathbf{m}_{\tilde{\mathbf{z}}}^\dagger \tag{13.46}$$

The normalizing quantity $S\left(1 - e^{-2\mu d_o}\right)$ that divides the autocorrelation $r_{\tilde{v}}(\xi)$ and the PSD $G_{\tilde{v}}(f)$ in Equations (13.44) and Equation (13.45) is equal to $r_{\tilde{v}}(0) = \int G_{\tilde{v}}(f)df$. Therefore, the maximum of the normalized autocorrelation is unity, and the area under the normalized PSD is unity. The complex envelope samples $\tilde{\mathbf{z}}(t)$ have the complex normal distribution $\mathrm{CN}(\mathbf{m}_{\tilde{\mathbf{z}}}, \mathbf{C}_{\tilde{\mathbf{z}}})$, which is defined in Equation (13.30). The mean vector and covariance matrix are given in Equations (13.43) and (13.46), but we repeat them below for comparison with Equation (13.29):

$$\mathbf{m}_{\tilde{\mathbf{z}}} = e^{-\mu d_o} \sqrt{S}\, e^{j\theta} \mathbf{a} \quad \text{(with scattering)} \tag{13.47}$$

$$\mathbf{C}_{\tilde{\mathbf{z}}} = S\left[\mathbf{B} \circ \left(\mathbf{a}\mathbf{a}^\dagger\right) - e^{-2\mu d_o} \mathbf{a}\mathbf{a}^\dagger\right] + \sigma_{\tilde{w}}^2 \mathbf{I} \quad \text{(with scattering)} \tag{13.48}$$

© 2005 by Chapman & Hall/CRC

Note that the scattering is negligible if $d_o \ll 1/(2\mu)$, in which case $e^{-2\mu d_o} \approx 1$ and $\Omega \approx 0$. Then most of the signal energy is in the mean, with $\mathbf{B} \approx \mathbf{1}\mathbf{1}^T$ and $\gamma_{mn} \approx 1$ in Equation (13.40), since $\nu(\rho_{mn}) < 2\mu$. For larger values of the source range d_o, more of the signal energy is scattered, and $\mathbf{B}$ may deviate from $\mathbf{1}\mathbf{1}^T$ (and $\gamma_{mn} < 1$ for $m \neq n$) due to coherence losses between the sensors. At full saturation ($\Omega = 1$), $\mathbf{B} = \mathbf{\Gamma}$.

The scattering model in Equation (13.41) may be formulated as multiplicative noise on the steering vector.

$$\tilde{\mathbf{z}}(t) = \sqrt{S}\, e^{j\theta}\, \mathbf{a} \circ \left[e^{-\mu d_o}\, \mathbf{1} + \frac{\tilde{\mathbf{v}}(t)}{\sqrt{S}} \right] + \tilde{\mathbf{w}}(t) \triangleq \sqrt{S}\, e^{j\theta}\, (\mathbf{a} \circ \tilde{\mathbf{u}}(t)) + \tilde{\mathbf{w}}(t) \tag{13.49}$$

The multiplicative noise process $\tilde{\mathbf{u}}(t)$ is complex normal with $\mathbf{m}_{\tilde{\mathbf{u}}} = E\{\tilde{\mathbf{u}}(t)\} = e^{-\mu d_o}\, \mathbf{1}$ and $E\{\tilde{\mathbf{u}}(t)\, \tilde{\mathbf{u}}(t)^\dagger\} = \mathbf{B}$, so the covariance matrix is $\mathbf{C}_{\tilde{\mathbf{u}}} = \mathbf{B} - e^{-2\mu d_o}\, \mathbf{1}\mathbf{1}^T = \Omega\mathbf{\Gamma}$, where $\mathbf{\Gamma}$ has elements γ_{mn} in Equation (13.40). The mean vector and covariance matrix in Equations (13.47) and (13.48) may be represented as $\mathbf{m}_{\tilde{\mathbf{z}}} = \sqrt{S}\, e^{j\theta}\, (\mathbf{a} \circ \mathbf{m}_{\tilde{\mathbf{u}}})$ and $\mathbf{C}_{\tilde{\mathbf{z}}} = S[(\mathbf{a}\mathbf{a}^\dagger) \circ \mathbf{C}_{\tilde{\mathbf{u}}}] + \sigma_w^2\, \mathbf{I}$.

13.2.4 Model for Extinction Coefficients

During the past several decades, considerable effort has been devoted to the modeling of wave propagation through random media. Theoretical models have been developed for the extinction coefficients of the first and second moments, μ and $\nu(\rho)$, along nearly line-of-sight paths. For general background, we refer the reader to Refs [5,10,27,30]. Here, we consider some specific results relevant to turbulence effects on aeroacoustic arrays.

The extent that scattering affects array performance depends on many factors, including the wavelength of the sound, the propagation distance from the source to the sensor array, the spacing between the sensors, the strength of the turbulence (as characterized by the variance of the temperature and wind-velocity fluctuations), and the size range of the turbulent eddies. Turbulence in the atmosphere near the ground spans a vast range of spatial scales, from millimeters to hundreds of meters. If the sensor spacing ρ is small compared with the size ℓ of the smallest eddies (a case highly relevant to optics but not low-frequency acoustics), $\nu(\rho)$ is proportional to $k^2\rho^2$, where $k = \omega/c_0$ is the wavenumber of the sound and c_0 the ambient sound speed [27]. In this situation, the loss in coherence between sensors results entirely from turbulence-induced variability in the AOA. Of greater practical importance in acoustics are situations where $\rho \gg \ell$. The spacing ρ may be smaller or larger than $\mathcal{L}$, the size of the largest eddies.

When $\rho \gg \ell$ and $\rho \ll \mathcal{L}$, the sensor spacing resides in the *inertial subrange* of the turbulence [5]. Because the strength of turbulence increases with the size of the eddies, this case has qualitative similarities to $\rho \ll \ell$. The wavefronts impinging on the array have a roughly constant AOA over the aperture and the apparent bearing of the source varies randomly about the actual bearing. Increasing the separation between sensors can dramatically decrease the coherence. In contrast, when $\rho \gg \mathcal{L}$ is large, the wavefront distortions induced by the turbulence produce nearly uncorrelated signal variations at the sensors. In this case, further increasing separation does not affect coherence: it is "saturated" at a value determined by the strength of the turbulence and, therefore, has an effect similar to additive, uncorrelated noise. These two extreme cases are illustrated in Figure 13.3. The resulting behavior of $\nu(\rho)$ and B_{mn} [Equation (13.42)] are shown in Figure 13.4.

The general results for the extinction coefficients of a spherically propagating wave, derived with the parabolic (narrow-angle) and Markov approximations, and assuming $\rho \gg \ell$, are [Ref. [10]: Equations (7.60) and (7.71); Ref. [30]: Equations (20)–(28)]:

$$\mu = \frac{\pi^2 k^2}{2} \int_0^\infty dK_\perp K_\perp \Phi_{\text{eff}}(K_\parallel = 0, K_\perp) = k^2 \sigma_{\text{eff}}^2 \mathcal{L}_{\text{eff}}/4 \tag{13.50}$$

© 2005 by Chapman & Hall/CRC

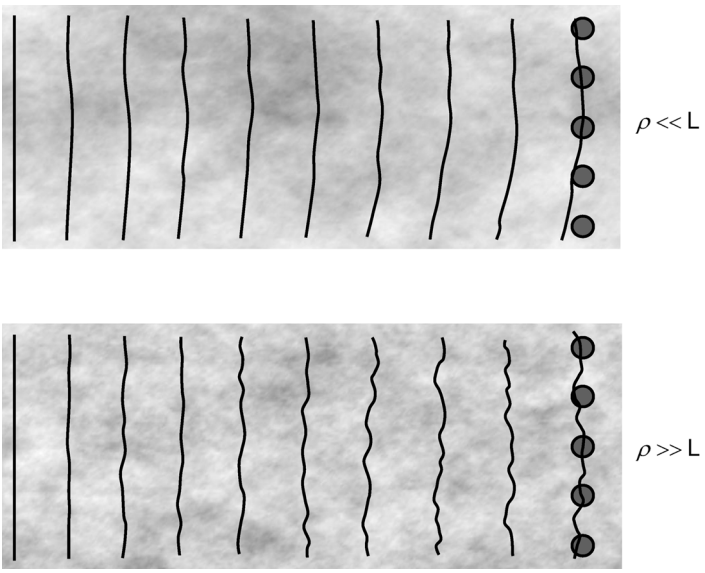


Figure 13.3. Turbulence-induced distortions of acoustic wavefronts impinging on an array. The wavefronts are initially smooth (left) and become progressively more distorted until they arrive at the array (right). Top: sensor separations within the inertial subrange of the turbulence ($\rho \gg \ell$ and $\rho \ll \mathcal{L}$). The wavefronts are fairly smooth but the AOA (and therefore the apparent source bearing) varies. Bottom: sensor separations much larger than the scale of the largest turbulent eddies ($\rho \gg \mathcal{L}$). The wavefronts have a very rough appearance and the effect of the scattering is similar to uncorrelated noise.

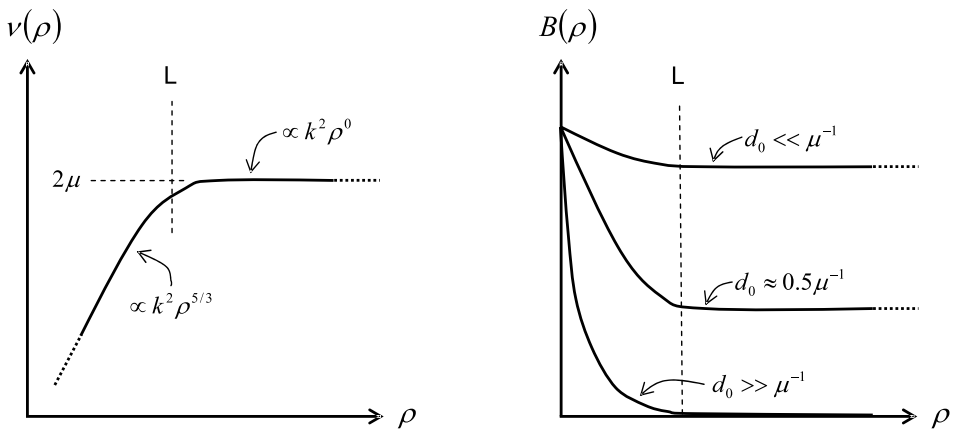


Figure 13.4. Left: characteristic behavior of the second-moment extinction coefficient $\nu(\rho)$. It initially increases with increasing sensor separation ρ, and then saturates at a fixed value 2μ (where μ is the first-moment extinction coefficient) when ρ is large compared with the size of the largest turbulent eddies. Right: resulting behavior of the total signal coherence B_{mn}, Equation (13.42), for several values of the propagation distance d_o.

$$\nu(\rho) = \pi^2 k^2 \int_0^1 \mathrm{d}t \int_0^\infty \mathrm{d}K_\perp K_\perp [1 - J_0(K_\perp \rho t)] \Phi_{\mathrm{eff}}(K_\parallel = 0, K_\perp) \tag{13.51}$$

in which J_0 is the zeroth-order Bessel function of the first kind and $\mathbf{K} = \mathbf{K}_\parallel + \mathbf{K}_\perp$ is the turbulence wavenumber vector decomposed into components parallel and perpendicular to the propagation path.

© 2005 by Chapman & Hall/CRC

The quantities $\Phi_{\text{eff}}(\mathbf{K})$, σ_{eff}, and $\mathcal{L}_{\text{eff}}$ are the effective turbulence spectrum, effective variance, and effective integral length scale. (The integral length scale is a quantitative measure of the size of the largest eddies.) The spectrum is defined as

$$\Phi_{\text{eff}}(\mathbf{K}) = \frac{\Phi_T(\mathbf{K})}{T_0^2} + \frac{4\Phi_\upsilon(\mathbf{K})}{c_0^2} \tag{13.52}$$

where T_0 is the ambient temperature, and the subscripts T and υ indicate the temperature and wind-velocity fields respectively. The definition of the effective variance is the same, except with σ^2 replacing $\Phi(\mathbf{K})$. The effective integral length scale is defined as

$$\mathcal{L}_{\text{eff}} = \frac{1}{\sigma_{\text{eff}}}\left(\mathcal{L}_T\frac{\sigma_T^2}{T_0^2} + \mathcal{L}_\upsilon\frac{4\sigma_\upsilon^2}{c_0^2}\right) \tag{13.53}$$

For the case $\rho/\mathcal{L}_{\text{eff}} \gg 1$, the contribution from the term in Equation (13.51) involving the Bessel function is small and one has $\nu(\rho) \to 2\mu$, as anticipated from the discussion after Equation (13.40). When $\rho/\mathcal{L}_{\text{eff}} \ll 1$, the inertial-subrange properties of the turbulence come into play and one finds [Ref. [10], Equation (7.87)]

$$\nu(\rho) = 0.137\left(\frac{C_T^2}{T_0^2} + \frac{22}{3}\frac{C_\upsilon^2}{c_0^2}\right)k^2\rho^{5/3} \tag{13.54}$$

where C_T^2 and C_υ^2 are the *structure-function parameters* for the temperature and wind fields respectively. The structure-function parameters represent the strength of the turbulence in the inertial subrange.

Note that the extinction coefficients for both moments depend quadratically on the frequency of the tone, regardless of the separation between the sensors. The quantities μ, C_T^2, C_υ^2, and $\mathcal{L}_{\text{eff}}$ each depend strongly on atmospheric conditions. Table 13.1 provides estimated values for typical atmospheric conditions based on the turbulence models in Refs. [11,24]. These calculations were performed for a propagation path height of 2 m.

It is evident from Table 13.1 that the entire range of saturation parameter values from $\Omega \approx 0$ to $\Omega \approx 1$ may be encountered in aeroacoustic applications, which typically have source ranges from meters to kilometers. Also, saturation occurs at distances several times closer to the source in sunny

Table 13.1. Modeled turbulence quantities and inverse extinction coefficients for various atmospheric conditions. The atmospheric conditions are described quantitatively in [24]. The second and third columns give the inverse extinction coefficients at 50 Hz and 200 Hz, respectively. These values indicate the distance at which random fluctuations in the complex signal become strong. The fourth and fifth columns represent the relative contributions of temperature and wind fluctuations to the field coherence. The sixth column is the effective integral length scale for the scattered sound field; at sensor separations greater than this value, the coherence is "saturated"

Atmospheric condition	μ^{-1} (m) at 50 Hz	μ^{-1} (m) at 200 Hz	C_T^2/T_0^2 ($\text{m}^{-2/3}$)	$(22/3)C_v^2/C_0^2$ ($\text{m}^{-2/3}$)	$\mathcal{L}_{\text{eff}}$ (m)
Mostly sunny, light wind	990	62	2.0×10^{-5}	8.0×10^{-6}	100
Mostly sunny, moderate wind	980	61	7.6×10^{-6}	2.8×10^{-5}	91
Mostly sunny, strong wind	950	59	2.4×10^{-6}	1.3×10^{-4}	55
Mostly cloudy, light wind	2900	180	1.5×10^{-6}	4.4×10^{-6}	110
Mostly cloudy, moderate wind	2800	180	4.5×10^{-7}	2.4×10^{-5}	75
Mostly cloudy, strong wind	2600	1160	1.1×10^{-7}	11.2×10^{-4}	28

© 2005 by Chapman & Hall/CRC

conditions than in cloudy ones. In a typical scenario in aeroacoustics involving a sensor standoff distance of several hundred meters, saturation will be small only for frequencies of about 100 Hz and lower. At frequencies above 200 Hz or so, the signal is generally saturated and random fluctuations dominate.

Based on the values for C_T^2 and C_v^2 in Table 13.1, coherence of signals is determined primarily by wind-velocity fluctuations (as opposed to temperature), except for mostly sunny, light wind conditions. It may at first seem a contradiction that the first-moment extinction coefficient μ is determined mainly by cloud cover (which affects solar heating of the ground), as opposed to the wind speed. Indeed, the source distance d_o at which a given value of Ω is obtain is several times longer in cloudy conditions than in sunny ones. This can be understood from the fact that cloud cover damps strong thermal plumes (such as those used by hang gliders and seagulls to stay aloft), which are responsible for wind-velocity fluctuations that strongly affect acoustic signals.

Interestingly, the effective integral length scale for the sound field usually takes on a value intermediate between the microphone separations within small arrays (less than or equal to 1 m) and the spacing between typical network nodes (which may be 100 m or more). As a result, high coherence can be expected within small arrays. However, coherence between nodes in a widely spaced network can be quite small, particularly at frequencies above 200 Hz or so.

Figure 13.5 illustrates the coherence of the scattered signals, γ_{mn} in Equation (13.40), as a function of the sensor separation ρ. The extinction coefficient in Equation (13.54) is computed at frequency $f = 50$ Hz and source range $d_o = 1500$ m, with mostly sunny, light wind conditions from Table 13.1, so $\Omega = 0.95$. Note that the coherence is nearly perfect for sensor separations $\rho < 1$ m; the coherence then declines steeply for larger separations.

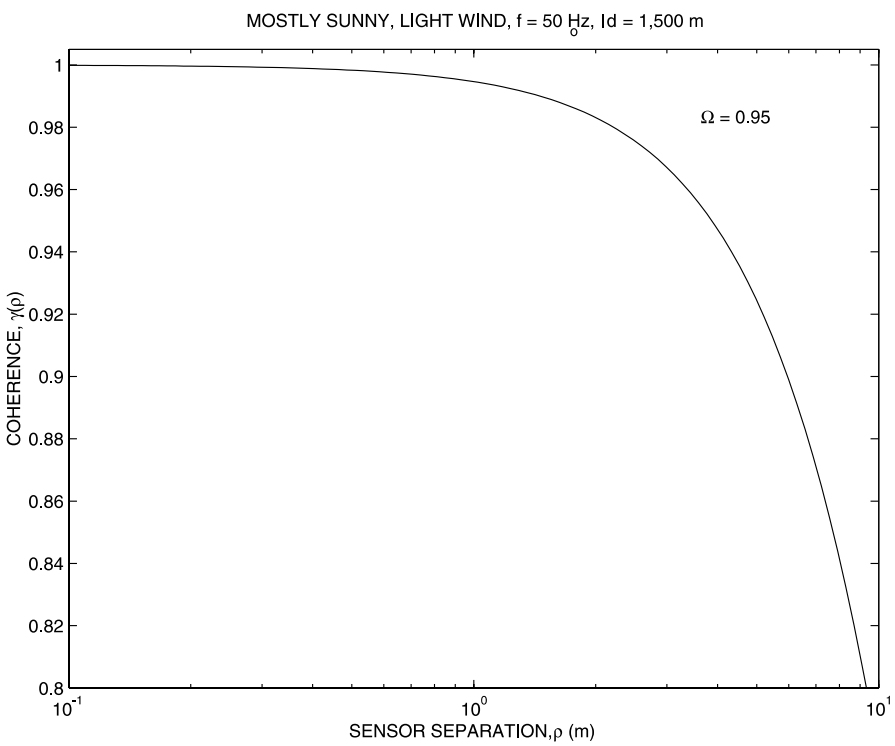


Figure 13.5. Evaluation of the coherence of the scattered signals at sensors with separation ρ, using $f = 50$ Hz, $d_o = 1500$ m, mostly sunny, light wind conditions (Table 13.1), $\nu(\rho)$ is computed with Equation (13.54), and the coherence $\gamma(\rho)$ is computed with Equation (13.40).

© 2005 by Chapman & Hall/CRC

13.2.5 Multiple Frequencies and Sources

The model in Equation (13.49) is for a single source that emits a single frequency, $\omega = 2\pi f_o$ rad/s. The complex envelope processing in Equation (13.2) and Figure 13.2 is a function of the source frequency. We can extend the model in Equation (13.49) to the case of K sources that emit tones at L frequencies $\omega_1, \ldots, \omega_L$, as follows:

$$\tilde{\mathbf{z}}(iT_s; \omega_l) = \sum_{k=1}^{K} \sqrt{S_k(\omega_l)}\, e^{j\theta_{k,l}} [\mathbf{a}_k(\omega_l) \circ \tilde{\mathbf{u}}_k(iT_s; \omega_l)] + \tilde{\mathbf{w}}(iT_s; \omega_l) \quad \begin{matrix} i = 1, \ldots, T \\ l = 1, \ldots, L \end{matrix} \tag{13.55}$$

$$= \left\{ [\mathbf{a}_1(\omega_l) \ldots \mathbf{a}_K(\omega_l)] \circ [\tilde{\mathbf{u}}_1(iT_s; \omega_l) \ldots \tilde{\mathbf{u}}_K(iT_s; \omega_l)] \right\} \begin{bmatrix} \sqrt{S_1(\omega_l)} e^{j\theta_{1,l}} \\ \vdots \\ \sqrt{S_K(\omega_l)} e^{j\theta_{K,l}} \end{bmatrix} + \tilde{\mathbf{w}}(iT_s; \omega_l)$$

$$\triangleq \left(\mathbf{A}(\omega_l) \circ \tilde{\mathbf{U}}(iT_s; \omega_l) \right) \tilde{\mathbf{p}}(\omega_l) + \tilde{\mathbf{w}}(iT_s \omega_l) \tag{13.56}$$

In Equation (13.55), $S_k(\omega_l)$ is the average power of source k at frequency ω_l, $\mathbf{a}_k(\omega_l)$ is the steering vector for source k at frequency ω_l as in Equation (13.20), $\tilde{\mathbf{u}}_k(iT_s; \omega_l)$ is the scattering of source k at frequency ω_l at time sample i, and T is the number of time samples. In Equation (13.56), the steering vector matrices $\mathbf{A}(\omega_l)$, the scattering matrices $\tilde{\mathbf{U}}(iT_s; \omega_l)$, and the source amplitude vectors $\tilde{\mathbf{p}}(\omega_l)$ for $l = 1, \ldots, L$ and $i = 1, \ldots, T$, are defined by the context. If the sample spacing T_s is chosen appropriately, then the samples at a given frequency ω_l are independent in time. We will also model the scattered signals at different frequencies as independent. Cross-frequency coherence has been previously studied theoretically and experimentally, with Refs [8,31] presenting experimental studies in the atmosphere. However, models for cross-frequency coherence in the atmosphere are at a very preliminary stage. It may be possible to revise the assumption of independent scattering at different frequencies as better models become available.

The covariance matrix at frequency ω_l is, by extending the discussion following Equation(13.49),

$$\mathbf{C}_{\tilde{\mathbf{z}}}(\omega_l) = \sum_{k=1}^{K} S_k(\omega_l)\, \Omega_k(\omega_l) \left[\mathbf{\Gamma}_k(\omega_l) \circ \left(\mathbf{a}_k(\omega_l) \mathbf{a}_k(\omega_l)^{\dagger} \right) \right] + \sigma_{\tilde{w}}(\omega_l)^2\, \mathbf{I} \tag{13.57}$$

where the scattered signals from different sources are assumed to be independent. If we assume full saturation ($\Omega_k(\omega_l) = 1$) and negligible coherence loss across the array aperture ($\mathbf{\Gamma}_k(\omega_l) = \mathbf{1}\mathbf{1}^{\mathrm{T}}$), then the sensor signals in Equation (13.55) have zero mean, and the covariance matrix in Equation (13.57) reduces to the familiar correlation matrix of the form

$$\mathrm{R}_{\tilde{\mathbf{z}}}(0; \omega_l) = E\left\{ \tilde{\mathbf{z}}(iT_s; \omega_l)\, \tilde{\mathbf{z}}(iT_s; \omega_l)^{\dagger} \right\}$$

$$= \mathbf{A}(\omega_l) \mathrm{S}(\omega_l) \mathbf{A}(\omega_l)^{\dagger} + \sigma_{\tilde{w}}(\omega_l)^2 \mathrm{I} \quad (\Omega_k(\omega_l) = 1 \text{ and no coherence loss}) \tag{13.58}$$

where $\mathbf{S}(\omega_l)$ is a diagonal matrix with $S_1(\omega_l), \ldots, S_K(\omega_l)$ along the diagonal.[1]

13.3 Signal Processing

In this section, we discuss signal processing methods for aeroacoustic sensor networks. The signal processing takes into account the source and propagation models presented in the previous section, as well as minimization of the communication bandwidth between sensor nodes connected by a wireless link. We begin with angle of arrival (AOA) estimation using a single sensor array in Section 13.3.1. Then

[1] For the fully saturated case with no coherence loss, we can relax the assumption that the scattered signals from different sources are independent by replacing the diagonal matrix $\mathbf{S}(\omega_l)$ in Equation (13.58) with a positive semidefinite matrix with (m, n) element $\sqrt{S_m(\omega_l) S_n(\omega_l)}\, E\{\tilde{u}_m(iT_s; \omega_l) \tilde{u}_n(iT_s; \omega_l)^*\}$, where $\tilde{u}_m(iT_s; \omega_l)$ is the scattered signal for source m.

© 2005 by Chapman & Hall/CRC

we discuss source localization with multiple sensor arrays in Section 13.3.2, and we briefly describe implications for tracking, detection, and classification algorithms in Sections 13.3.3 and 13.3.4.

13.3.1 AOA Estimation

We discuss narrowband AOA estimation with scattering in Section 13.3.1.1, and then we discuss wideband AOA estimation without scattering in Section 13.3.1.2.

13.3.1.1 Narrowband AOA Estimation with Scattering

In this section, we review some performance analyses and algorithms that have been investigated for narrowband AOA estimation with scattering. Most of the methods are based on scattering models that are similar to the single-source model in Section 13.2.3 or the multiple-source model in Section 13.2.5 at a single frequency. Many of the references cited below are formulated for radio frequency (RF) channels, so the equivalent channel effect is caused by multipath propagation and Doppler. The models for the RF case are similar to those presented in Section 13.2.

Wilson [21] analyzed the Cramér–Rao bound (CRB) on AOA estimation for a single source using several models for atmospheric turbulence. Rayleigh signal fading was assumed. Collier and Wilson [22,23] extended the work to include unknown turbulence parameters in the CRB, along with the source AOA. Their CRB analysis provides insight into the combinations of atmospheric conditions, array geometry, and source location that are favorable for accurate AOA estimation. They note that refraction effects make it difficult to estimate the elevation angle accurately when the source and sensors are near the ground, so aeroacoustic sensor arrays are most effective for azimuth estimation.

Other researchers [32–40] have investigated the problem of imperfect spatial coherence in the context of narrowband AOA estimation. Paulraj and Kailath [32] presented a MUSIC algorithm that incorporates nonideal spatial coherence, assuming that the coherence losses are known. Song and Ritcey [33] provided maximum-likelihood (ML) methods for estimating the AOAs and the parameters in a coherence model. Gershman *et al.* [34] provided a procedure to jointly estimate the spatial coherence loss and the AOAs. Gershman and co-workers [35–38] studied stochastic and deterministic models for imperfect spatial coherence, and the performance of various AOA estimators was analyzed. Ghogho *et al.* [39] presented an algorithm for AOA estimation with multiple sources in the fully saturated case. Their algorithm exploits the Toeplitz structure of the $\mathbf{B}$ matrix in Equation (13.42) for a uniform linear array (ULA).

None of the Refs [32–39] handles the full range of scattering scenarios from weak ($\Omega = 0$) to strong ($\Omega = 1$). Fuks *et al.* [40] treat the case of Rician scattering on RF channels, so this approach does include the entire range from weak to strong scattering. Indeed, the "Rice factor" in the Rician fading model is related to the saturation parameter through $(1 - \Omega)/\Omega$. The main focus of Fuks *et al.* [40] is on CRBs for AOA estimation.

13.3.1.2 Wideband AOA Estimation without Scattering

Narrowband processing in the aeroacoustic context will limit the bandwidth to perhaps a few hertz, and the large fractional bandwidth encountered in aeroacoustics significantly complicates the array signal processing. A variety of methods are available for wideband AOA estimation, with varying complexity and applicability. Application of these to specific practical problems leads to a complicated task of appropriate procedure choice. We outline some of these methods and various tradeoffs, and describe some experimental results. Basic approaches include: classical delay-and-sum beamformer, incoherent averaging over narrowband spatial spectra, maximum likelihood (ML), coherent signal subspace methods, steered matrix techniques, spatial resampling (array interpolation), and frequency-invariant beamforming. Useful overviews include Boehme [41], and Van Trees [42]. Significant progress in this area has occurred in the previous 15 years or so; major earlier efforts include the underwater acoustics area, e.g. see Owsley [43].

© 2005 by Chapman & Hall/CRC

Using frequency decomposition at each sensor, we obtained the array data model in Equation (13.55). For our discussion of wideband AOA methods, we will ignore the scattering, and so assume the spatial covariance can be written as in Equation (13.58). Equation (13.58) may be interpreted as the covariance matrix of the Fourier-transformed (narrowband) observations of Equation (13.55). The noise is typically assumed to be Gaussian and spatially white, although generalizations to spatially correlated noise are also possible, which can be useful for modeling unknown spatial interference.

Working with an estimate $\hat{\mathbf{R}}_{\tilde{\mathbf{z}}}(0;\ \omega_l)$, we may apply covariance-based high resolution AOA estimators (MUSIC, MLE, etc.), although this results in many frequency-dependent angle estimates that must be associated in some way for each source. A simple approach is to sum the resulting narrowband spatial spectra, e.g. see [44]; this is referred to as *noncoherent averaging*. This approach has the advantages of straightforward extension of narrowband methods and relatively low complexity, but it can produce artifacts. And, noncoherent averaging requires that the SNRs after channelization be adequate to support the chosen narrow band AOA estimator; in effect the method does not take strong advantage of the wideband nature of the signal. However, loud harmonic sources can be processed in this manner with success.

A more general approach was first developed by Wang and Kaveh [45], based on the following additive composition of transformed narrowband covariance matrices:

$$\mathbf{R}_{\text{scm}}(\phi_i) = \sum_l \mathbf{T}(\phi_i,\ \omega_l)\, \mathbf{R}_{\tilde{\mathbf{z}}}(0;\ \omega_l)\, \mathbf{T}(\phi_i,\ \omega_l)^{\dagger} \tag{13.59}$$

where ϕ_i is the ith AOA. $\mathbf{R}_{\text{scm}}(\phi_i)$ is referred to as the *steered covariance matrix* or the *focused wideband covariance matrix*. The transformation matrix $\mathbf{T}(\phi_i,\ \omega_l)$, sometimes called the *focusing matrix*, can be viewed as selecting delays to coincide with delay-sum beamforming, so that the transformation depends on both AOA and frequency. Viewed in another way, the transformation matrix acts to align the signal subspaces, so that the resulting matrix $\mathbf{R}_{\text{scm}}(\phi_i)$ has a rank one contribution from a wideband source at angle ϕ_i. Now, narrowband covariance-based AOA estimation methods may be applied to the matrix $\mathbf{R}_{\text{scm}}(\phi_i)$. This approach is generally referred to as the *coherent subspace method* (CSM). The CSM has significant advantages: it can handle correlated sources (due to the averaging over frequencies), it averages over the entire source bandwidth, and has good statistical stability. On the other hand, it requires significant complexity and, as originally proposed, requires pre-estimation of the AOAs, which can lead to biased estimates [46]. (Valaee and Kabal [47] present an alternative formulation of focusing matrices for the CSM using a two-sided transformation, attempting to reduce the bias associated with the CSM.)

A major drawback to the CSM is the dependence of $\mathbf{T}$ on the the AOA. The most general form requires generation and eigendecomposition of $\mathbf{R}_{\text{scm}}(\phi_i)$ for each look angle; this is clearly undesirable from a computational standpoint.[2] The dependence of $\mathbf{T}$ on ϕ_i can be removed in some cases by incorporating spatial interpolation, thereby greatly reducing the complexity. The basic ideas are established by Krolik and Swingler in [48]; for an overview (including CSMs) see Krolik [49].

As an example, consider a ULA [48,49], with $d = \lambda_i/2$ spacing. In order to process over another wavelength choice λ_j ($\lambda_j > \lambda_i$), we could spatially interpolate the physical array to a virtual array with the desired spacing ($d_j = \lambda_j/2$). The spatial resampling approach adjusts the spatial sampling interval d as a function of source wavelength λ_j. The result is a simplification of Equation (13.59) to

$$\mathbf{R}_{\text{sr}} = \sum_l \mathbf{T}(\omega_l)\, \mathbf{R}_{\tilde{\mathbf{z}}}(0;\ \omega_l)\, \mathbf{T}(\omega_l)^{\dagger} \tag{13.60}$$

where the angular dependence is now removed. The resampling acts to align the signal subspace contributions over frequency, so that a single wideband source results in a rank one contribution to $\mathbf{R}_{\text{sr}}$. Note that the spatial resampling is implicit in Equation (13.60) via the matrices $\mathbf{T}(\omega_l)$. Conventional

[2] In their original work, Wang and Kaveh [45] relied on pre-estimates of the AOAs to lower the computational burden.

© 2005 by Chapman & Hall/CRC

narrowband AOA estimation methods may now be applied to $\mathbf{R}_{sr}$, and, in contrast to CSM, this operation is conducted once for all angles.

Extensions of [48] from ULAs to arbitrary array geometries can be undertaken, but the dependence on look angle returns, and the resulting complexity is then similar to the CSM approaches. To avoid this, Friedlander and Weiss [50] considered spatial interpolation of an arbitrary physical array to virtual arrays that are uniform and linear, thereby returning to a formulation like Equation (13.60). Doron *et al.* [51] developed a spatial interpolation method for forming a focused covariance matrix with arbitrary arrays. The formulation relies on a truncated series expansion of plane waves in polar coordinates. The array manifold vector is now separable, allowing focusing matrices that are not a function of angle. The specific case of a circular array leads to an FFT-based implementation that is appealing due to its relatively low complexity.

While the spatial resampling methods are clearly desirable from a complexity standpoint, experiments indicate that they break down as the fractional bandwidth grows (see the examples that follow). This depends on the particular method, and the original array geometry. This may be due to accumulated interpolation error, undersampling, and calibration error. As we have noted, and show in our examples, fractional bandwidths of interest in aeroacoustics may easily exceed 100%. Thus, the spatial resampling methods should be applied with some caution in cases of large fractional bandwidth.

Alternatives to the CSM approach are also available. Many of these methods incorporate time domain processing, and so may avoid the frequency decomposition (discrete fourier transform) associated with CSM. Buckley and Griffiths [52] and Agrawal and Prasad [53] have developed methods based on wideband correlation matrices. (The work of Agrawal and Prasad [53] generally relies on a white or near-white source spectrum assumption, and so might not be appropriate for harmonic sources.) Sivanand and co-workers [54–56] have shown that the CSM focusing can be achieved in the time domain, and treat the problem from a multichannel finite impulse response (FIR) filtering perspective. Another FIR-based method employs frequency-invariant beamforming, e.g. see Ward *et al.* [57] and references therein.

13.3.1.3 Performance Analysis and Wideband Beamforming

CRBs on wideband AOA estimation can be established using either a deterministic or random Gaussian source model, in additive Gaussian noise. The basic results were shown by Bangs [58]; see also Swingler [59]. The deterministic source case in (possibly colored) Gaussian noise is described by Kay [20]. Performance analysis of spatial resampling methods is considered by Friedlander and Weiss [50], who also provide CRBs, as well as a description of ML wideband AOA estimation.

These CRBs typically require known source statistics, apply to unbiased estimates, and assume no scattering, whereas prior spectrum knowledge is usually not available, and the above wideband methods may result in biased estimates. Nevertheless, the CRB provides a valuable fundamental performance bound.

Basic extensions of narrowband beamforming methods are reviewed by Van Trees [42, chapter 6], including delay-sum and wideband minimum variance distortionless response (MVDR) techniques. The CSM techniques also extend to wideband beamforming, e.g. see Yang and Kaveh [60].

13.3.1.4 AOA Experiments

Next, we highlight some experimental examples and results, based on extensive aeroacoustic experiments carried out since the early 1990s [3,61–66]. These experiments were designed to test wideband superresolution AOA estimation algorithms based on array apertures of a few meters or less. The arrays were typically only approximately calibrated, roughly operating in $[50, 250]$ Hz, primarily circular in geometry, and planar (on the ground). Testing focused on military vehicles, and low-flying rotary and fixed-wing aircraft, and ground truth was typically obtained from global positioning satellite (GPS) receivers on the sources.

© 2005 by Chapman & Hall/CRC

Early results showed that superresolution AOA estimates could be achieved at ranges of 1 to 2 km [61], depending on the various propagation conditions and source loudness, and that noncoherent summation of narrowband MUSIC spatial signatures significantly outperforms conventional wideband delay-sum beamforming [62]. When the sources had strong harmonic structure, it was a straightforward matter to select the spectral peaks for narrowband AOA estimation. These experiments also verified that a piecewise stationary assumption was valid over intervals approximately below 1 s, that the observed spatial coherence was good over apertures of a few meters or less, and that only rough calibration was required with relatively inexpensive microphones. Outlier AOA estimates were also observed, even in apparently high SNR and good propagation conditions. In some cases the outliers composed 10% of the AOA estimates, but these were infrequent enough that a robust tracking algorithm could reject them.

Tests of the CSM method (CSM-MUSIC) were conducted with diesel-engine vehicles exhibiting strong harmonic signatures [63], as well as turbine engines exhibiting broad, relatively flat spectral signatures [64]. The CSM-MUSIC approach was contrasted with noncoherent MUSIC. In both cases the M largest spectral bins were selected adaptively for each data block. CSM-MUSIC was implemented with a focusing matrix $\mathbf{T}$ diagonal. For harmonic source signatures, the noncoherent MUSIC method was shown to outperform CSM-MUSIC in many cases, generally depending on the observed narrowband SNRs [63]. On the other hand, the CSM-MUSIC method displays good statistical stability at a higher computational cost. And, inclusion of lower SNR frequency bins in noncoherent MUSIC can lead to artifacts in the resulting spatial spectrum.

For the broadband turbine source, the CSM-MUSIC approach generally performed better than noncoherent MUSIC, due to the ability of CSM to capture the broad spectral spread of the source energy [64]. Figure 13.6 depicts a typical experiment with a turbine vehicle, showing AOA estimates over a 250 s span, where the vehicle traverses approximately a ± 1 km path past the array. The largest $M = 20$ frequency bins were selected for each estimate. The AOA estimates (circles) are overlaid on GPS ground truth (solid line). The AOA estimators break down at the farthest ranges (the beginning and end

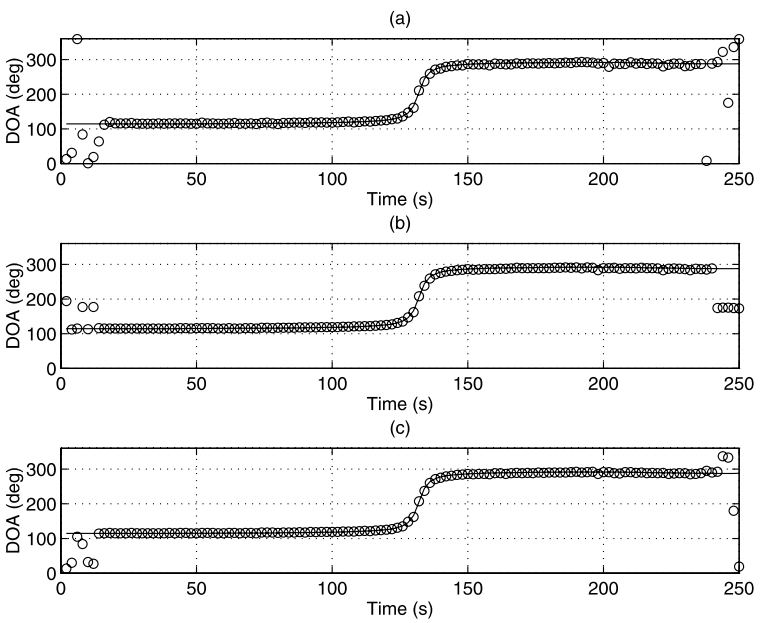


Figure 13.6. Experimental wideband AOA estimation over 250 s, covering a range of approximately ± 1 km. Three methods are depicted with M highest SNR frequency bins: (a) narrowband MUSIC ($M = 1$), (b) incoherent MUSIC ($M = 20$), and (c) CSM-MUSIC ($M = 20$). Solid lines depict GPS-derived AOA ground truth.

© 2005 by Chapman & Hall/CRC

of the data). Numerical comparison with the GPS-derived AOAs reveals CSM-MUSIC to have slightly lower mean-square error. While the three AOA estimators shown in Figure 13.6 for this single-source case have roughly the same performance, we emphasize that examination of the beam patterns reveals that the CSM-MUSIC method exhibits the best statistical stability and lower sidelobe behavior over the entire data set [64]. In addition, the CSM-MUSIC approach exhibited better performance in multiple-source testing.

Experiments with the spatial resampling approaches reveal that they require spatial oversampling to handle large fractional bandwidths [65,66]. For example, the array manifold interpolation (AMI) method of Doron *et al.* [51] was tested experimentally and via simulation using a 12-element uniform circular array. While the CSM-MUSIC approach was asymptotically efficient in simulation, the AMI technique did not achieve the CRB. The AMI algorithm performance degraded as the fractional bandwidth was increased for a fixed spatial sampling rate. While the AMI approach is appealing from a complexity standpoint, effective application of AMI requires careful attention to the fractional bandwidth, maximum source frequency, array aperture, and degree of oversampling. Generally, the AMI approach required higher spatial sampling when compared with CSM-type methods, and so AMI lost some of its potential complexity savings in both hardware and software.

13.3.2 Localization with Distributed Sensor Arrays

The previous subsection was concerned with AOA estimation using a single-sensor array. The (x, y) location of a source in the plane may be estimated efficiently using multiple-sensor arrays that are distributed over a wide area. We consider source localization in this section using a network of sensors that are placed in an "array of arrays" configuration, as illustrated in Figure 13.7. Each array contains local processing capability and a wireless communication link with a fusion center. A standard approach for estimating the source locations involves AOA estimation at the individual arrays, communication of the bearings to the fusion center, and triangulation of the bearing estimates at the fusion center (e.g. see Refs [67–71]). This approach is characterized by low communication bandwidth and low complexity, but the localization accuracy is generally inferior to the optimal solution in which the fusion center jointly processes all of the sensor data. The optimal solution requires high communication bandwidth, high processing complexity, and accurate time synchronization between arrays. The amount of improvement in localization accuracy that is enabled by greater communication bandwidth and processing complexity is dependent on the scenario, which we characterize in terms of the power spectra (and bandwidth) of the signals and noise at the sensors, the coherence between the source signals received at widely separated sensors, and the observation time (amount of data).

We have studied this scenario [16], where a framework is presented to identify situations that have the potential for improved localization accuracy relative to the standard bearings-only triangulation

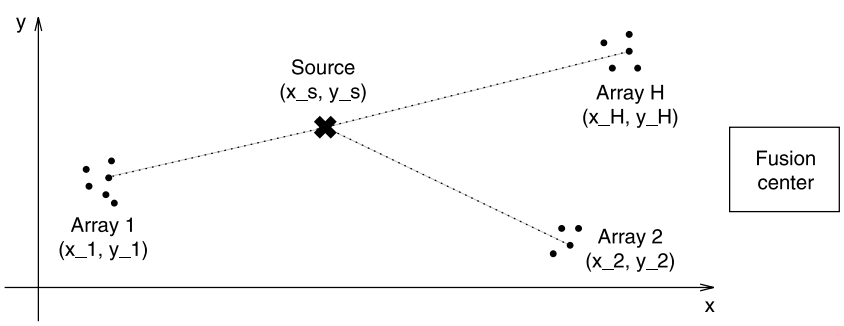


Figure 13.7. Geometry of nonmoving source location and an array of arrays. A communication link is available between each array and the fusion center. (Originally published in [16], ©2004 IEEE, reprinted with permission.)

© 2005 by Chapman & Hall/CRC

method. We proposed an algorithm that is bandwidth-efficient and nearly optimal that uses beamforming at small-aperture sensor arrays and time-delay estimation (TDE) between widely separated sensors. Accurate time-delay estimates using widely separated sensors are utilized to achieve improved localization accuracy relative to bearings-only triangulation, and the scattering of acoustic signals by the atmosphere significantly impacts the accuracy of TDE. We provide a detailed study of TDE with scattered signals that are *partially* coherent at widely-spaced sensors in [16]. Our results quantify the scenarios in which TDE is feasible as a function of signal coherence, SNR per sensor, fractional bandwidth of the signal, and time–bandwidth product of the observed data. The basic result is that, for a given SNR, fractional bandwidth, and time–bandwidth product, there exists a "threshold coherence" value that must be exceeded in order for TDE to achieve the CRB. The analysis is based on Ziv–Zakai bounds for TDE, expanding upon the results in [72,73]. Time synchronization is required between the arrays for TDE.

Previous work on source localization with aeroacoustic arrays has focused on AOA estimation with a *single* array, e.g. [61–66,74,75], as discussed in Section 13.3.1. The problem of imperfect spatial coherence in the context of narrowband angle-of-arrival estimation with a single array was studied in [21], [22,23], [32–40], as discussed in Section 3.1.1. The problem of decentralized array processing was studied in Refs [76,77]. Wax and Kailath [76] presented subspace algorithms for narrowband signals and distributed arrays, assuming perfect spatial coherence across each array but neglecting any spatial coherence that may exist between arrays. Stoica *et al.* [77] considered ML AOA estimation with a large, perfectly coherent array that is partitioned into subarrays. Weinstein [78] presented performance analysis for pairwise processing of the wideband sensor signals from a single array, and he showed that pairwise processing is nearly optimal when the SNR is high. Moses and Patterson [79] studied autocalibration of sensor arrays, where for aeroacoustic arrays the loss of signal coherence at widely separated sensors will impact the performance of autocalibration.

The results in [16] are distinguished from those cited in the previous paragraph in that the primary focus is a performance analysis that explicitly models partial spatial coherence in the signals at *different* sensor arrays in an array of arrays configuration, along with an analysis of decentralized processing schemes for this model. The previous studies have considered wideband processing of aeroacoustic signals using a single array with perfect spatial coherence [61–66,74,75], imperfect spatial coherence across a single-array aperture [21–23,32–40], and decentralized processing with either zero coherence between distributed arrays [76] or full coherence between all sensors [77,78]. We summarize the key results from [16] in Sections 13.3.2.1–13.3.2.3.

Source localization using the method of travel-time tomography is described in Refs [80,81]. In this type of tomography, TDEs are formed by cross-correlating signals from widely spaced sensors. The TDEs are incorporated into a general inverse procedure that provides information on the atmospheric wind and temperature fields in addition to the source location. The tomography thereby adapts to time-delay shifts that result from the intervening atmospheric structure.

Ferguson [82] describes localization of small-arms fire using the near-field wavefront curvature. The range and bearing of the source are estimated from two adjacent sensors. Ferguson's experimental results clearly illustrate random localization errors induced by atmospheric turbulence. In a separate article, Ferguson [83] discusses time-scale compression to compensate TDEs for differential Doppler resulting from fast-moving sources.

13.3.2.1 Model for Array of Arrays

Our model for the array of arrays scenario in Figure 13.7 is a wideband extension of the single-array, narrowband model in Section 13.2. Our array of arrays model includes two key assumptions:

1. The distance from the source to each array is sufficiently large so that the signals are fully saturated, i.e. $\Omega^{(h)}(\omega) \approx 1$ for $h = 1, \ldots, H$ and all ω. Therefore, according to the model in Section 13.2.3, the sensor signals have zero mean.

© 2005 by Chapman & Hall/CRC

2. Each array aperture is sufficiently small so that the coherence loss is negligible between sensor pairs in the array. For the example in Figure 13.5, this approximation is valid for array apertures less than 1 m.

It may be useful to relax these assumptions in order to consider the effects of nonzero mean signals and coherence losses across individual arrays. However, these assumptions allow us to focus on the impact of coherence losses in the signals at *different* arrays.

As in Section 13.2.1, we let (x_s, y_s) denote the coordinates of a single nonmoving source, and we consider H arrays that are distributed in the same plane, as illustrated in Figure 13.7. Each array $h \in \{1, \ldots, H\}$ contains N_h sensors and has a reference sensor located at coordinates (x_h, y_h). The location of sensor $n \in \{1, \ldots, N_h\}$ is at $(x_h + \Delta x_{hn},\ y_h + \Delta y_{hn})$, where $(\Delta x_{hn},\ \Delta y_{hn})$ is the relative location with respect to the reference sensor. If c is the speed of propagation, then the propagation time from the source to the reference sensor on array h is

$$\tau_h = \frac{d_h}{c} = \frac{1}{c}\left[(x_s - x_h)^2 + (y_s - y_h)^2\right]^{1/2} \tag{13.61}$$

where d_h is the distance from the source to array h, as in Equation (13.5). We model the wavefronts over individual array apertures as perfectly coherent plane waves; so, in the far-field approximation, the propagation time from the source to sensor n on array h is expressed by $\tau_h + \tau_{hn}$, where

$$\tau_{hn} \approx -\frac{1}{c}\left(\frac{x_s - x_h}{d_h}\Delta x_{hn} + \frac{y_s - y_h}{d_h}\Delta y_{hn}\right) = -\frac{1}{c}\left[(\cos\ \phi_h)\Delta x_{hn} + (\sin\ \phi_h)\Delta y_{hn}\right] \tag{13.62}$$

is the propagation time from the reference sensor on array h to sensor n on array h, and ϕ_h is the bearing of the source with respect to array h. Note that while the far-field approximation of Equation (13.62) is reasonable over individual array apertures, the wavefront curvature that is inherent in Equation (13.61) must be retained in order to model wide separations between arrays.

The time signal received at sensor n on array h due to the source will be denoted as $s_h(t - \tau_h - \tau_{hn})$, where the vector $\mathbf{s}(t) = [s_1(t), \ldots, s_H(t)]^{\mathrm{T}}$ contains the signals received at the reference sensors on the H arrays. The elements of $\mathbf{s}(t)$ are modeled as real-valued, continuous-time, zero-mean, jointly wide-sense stationary, Gaussian random processes with $-\infty < t < \infty$. These processes are fully specified by the $H \times H$ cross-correlation matrix

$$\mathbf{R}_s(\xi) = E\{\mathbf{s}(t + \xi)\,\mathbf{s}(t)^{\mathrm{T}}\} \tag{13.63}$$

The $(g,\ h)$ element in Equation (13.63) is the cross-correlation function

$$r_{s,gh}(\xi) = E\{s_g(t + \xi)\,s_h(t)\} \tag{13.64}$$

between the signals received at arrays g and h. The correlation functions (13.63) and (13.64) are equivalently characterized by their Fourier transforms, which are the CSD functions in Equation (13.65) and a CSD matrix in Equation (13.66):

$$G_{s,gh}(\omega) = \mathcal{F}\{r_{s,gh}(\xi)\} = \int_{-\infty}^{\infty} r_{s,gh}(\xi)\ \exp(-\mathrm{j}\omega\xi)\,\mathrm{d}\xi \tag{13.65}$$

$$\mathbf{G}_s(\omega) = \mathcal{F}\{\mathbf{R}_s(\xi)\} \tag{13.66}$$

© 2005 by Chapman & Hall/CRC

The diagonal elements $G_{s,hh}(\omega)$ of Equation (13.66) are the PSD functions of the signals $s_h(t)$, and hence they describe the distribution of average signal power with frequency. The model allows the PSD to vary from one array to another to reflect differences in transmission loss and source aspect angle.

The off-diagonal elements of Equation (13.66), $G_{s,gh}(\omega)$, are the CSD functions for the signals $s_g(t)$ and $s_h(t)$ received at distinct arrays $g \neq h$. In general, the CSD functions have the form

$$G_{s,gh}(\omega) = \gamma_{s,gh}(\omega)\left[G_{s,gg}(\omega)G_{s,hh}(\omega)\right]^{1/2} \tag{13.67}$$

where $\gamma_{s,gh}(\omega)$ is the *spectral coherence function* for the signals, which has the property $0 \leq |\gamma_{s,gh}(\omega)| \leq 1$. Coherence magnitude $|\gamma_{s,gh}(\omega)| = 1$ corresponds to perfect correlation between the signals at arrays g and h, while the partially coherent case $|\gamma_{s,gh}(\omega)| < 1$ models random scattering in the propagation paths from the source to arrays g and h. Note that our assumption of perfect spatial coherence across individual arrays implies that the scattering has negligible impact on the intra-array delays τ_{hn} in Equation (13.62) and the bearings $\phi_1, \ldots, \phi_H$. The coherence $\gamma_{s,gh}(\omega)$ in Equation (13.67) is an extension of the narrowband, short-baseline coherence γ_{mn} in Equation (13.39). However, the relation to extinction coefficients in Equation (13.40) is not necessarily valid for very large sensor separations.

The signal received at sensor n on array h is the delayed source signal plus noise.

$$z_{hn}(t) = s_h(t - \tau_h - \tau_{hn}) + w_{hn}(t) \tag{13.68}$$

where the noise signals $w_{hn}(t)$ are modeled as real-valued, continuous-time, zero-mean, jointly wide-sense stationary, Gaussian random processes that are mutually uncorrelated at distinct sensors, and are uncorrelated from the signals. That is, the noise correlation properties are

$$E\{w_{gm}(t+\xi)w_{hn}(t)\} = r_w(\xi)\,\delta_{gh}\delta_{mn} \quad \text{and} \quad E\{w_{gm}(t+\xi)s_h(t)\} = 0 \tag{13.69}$$

where $r_w(\xi)$ is the noise autocorrelation function, and the noise PSD is $G_w(\omega) = \mathcal{F}\{r_w(\xi)\}$. We then collect the observations at each array h into $N_h \times 1$ vectors $\mathbf{z}_h(t) = [z_{h1}(t), \ldots, z_{h,N_h}(t)]^{\mathrm{T}}$ for $h = 1, \ldots, H$, and we further collect the observations from the H arrays into a vector

$$\mathbf{Z}(t) = \left[\mathbf{z}_1(t)^{\mathrm{T}} \quad \ldots \quad \mathbf{z}_H(t)^{\mathrm{T}}\right]^{\mathrm{T}}. \tag{13.70}$$

The elements of $\mathbf{Z}(t)$ in Equation (13.70) are zero-mean, jointly wide-sense stationary, Gaussian random processes. We can express the CSD matrix of $\mathbf{Z}(t)$ in a convenient form with the following definitions. We denote the array steering vector for array h at frequency ω as

$$\mathbf{a}^{(h)}(\omega) = \begin{bmatrix} \exp(-\mathrm{j}\omega\tau_{h1}) \\ \vdots \\ \exp(-\mathrm{j}\omega\tau_{h,N_h}) \end{bmatrix} = \begin{bmatrix} \exp\left[\mathrm{j}\frac{\omega}{c}\left[(\cos\phi_h)\Delta x_{h1} + (\sin\phi_h)\Delta y_{h1}\right]\right] \\ \vdots \\ \exp\left[\mathrm{j}\frac{\omega}{c}\left[(\cos\phi_h)\Delta x_{h,N_h} + (\sin\phi_h)\Delta y_{h,N_h}\right]\right] \end{bmatrix} \tag{13.71}$$

using τ_{hn} from Equation (13.62) and assuming that the sensors have omnidirectional response. Let us define the relative time delay of the signal at arrays g and h as

$$D_{gh} = \tau_g - \tau_h \tag{13.72}$$

© 2005 by Chapman & Hall/CRC

where τ_h is defined in Equation (13.61). Then the CSD matrix of $\mathbf{Z}(t)$ in Equation (13.70) has the form

$$\mathbf{G}_{\mathbf{Z}}(\omega) = \begin{bmatrix} \mathbf{a}^{(1)}(\omega)\mathbf{a}^{(1)}(\omega)^{\dagger} G_{s,11}(\omega) & \cdots & \mathbf{a}^{(1)}(\omega)\mathbf{a}^{(H)}(\omega)^{\dagger} \exp(-\mathrm{j}\omega D_{1H}) G_{s,1H}(\omega) \\ \vdots & \ddots & \vdots \\ \mathbf{a}^{(H)}(\omega)\mathbf{a}^{(1)}(\omega)^{\dagger} \exp(+\mathrm{j}\omega D_{1H}) G_{s,1H}(\omega)^{*} & \cdots & \mathbf{a}^{(H)}(\omega)\mathbf{a}^{(H)}(\omega)^{\dagger} G_{s,HH}(\omega) \end{bmatrix} + G_w(\omega)\mathbf{I} \tag{13.73}$$

Recall that the source CSD functions $G_{s,gh}(\omega)$ in Equation (13.73) depend on the signal PSDs and spectral coherence $\gamma_{s,gh}(\omega)$ according to Equation (13.67). Note that Equation (13.73) depends on the source location parameters (x_s, y_s) through the bearings ϕ_h in $\mathbf{a}^{(h)}(\omega)$ and the pairwise time-delay differences D_{gh}.

13.3.2.2 CRBs and Examples

The problem of interest is estimation of the source location parameter vector $\boldsymbol{\Theta} = [x_s, y_s]^{\mathrm{T}}$ using T independent samples of the sensor signals $\mathbf{Z}(0), \mathbf{Z}(T_s), \ldots, \mathbf{Z}((T-1)T_s)$, where T_s is the sampling period. The total observation time is $\mathcal{T} = TT_s$, and the sampling rate is $f_s = 1/T_s$ and $\omega_s = 2\pi f_s$. We will assume that the continuous-time random processes $\mathbf{Z}(t)$ are band-limited, and that the sampling rate f_s is greater than twice the bandwidth of the processes. Then it has been shown [84,85] that the Fisher information matrix (FIM) $\mathbf{J}$ for the parameters $\boldsymbol{\Theta}$ based on the samples $\mathbf{Z}(0)$, $\mathbf{Z}(T_s), \ldots, \mathbf{Z}((T-1)T_s)$ has elements

$$J_{ij} = \frac{\mathcal{T}}{4\pi} \int_0^{\omega_s} \mathrm{tr}\left\{ \frac{\partial \mathbf{G}_{\mathbf{Z}}(\omega)}{\partial \theta_i} \mathbf{G}_{\mathbf{Z}}(\omega)^{-1} \frac{\partial \mathbf{G}_{\mathbf{Z}}(\omega)}{\partial \theta_j} \mathbf{G}_{\mathbf{Z}}(\omega)^{-1} \right\} \mathrm{d}\omega, \quad i, j = 1, 2 \tag{13.74}$$

where "tr" denotes the trace of the matrix. The CRB matrix $\mathbf{C} = \mathbf{J}^{-1}$ then has the property that the covariance matrix of any unbiased estimator $\hat{\boldsymbol{\Theta}}$ satisfies $\mathrm{Cov}(\hat{\boldsymbol{\Theta}}) - \mathbf{C} \geq \mathbf{0}$, where $\geq \mathbf{0}$ means that $\mathrm{Cov}(\hat{\boldsymbol{\Theta}}) - \mathbf{C}$ is positive semidefinite. Equation (13.74) provides a convenient way to compute the FIM for the array of arrays model as a function of the signal coherence between distributed arrays, the signal and noise bandwidth and power spectra, and the sensor placement geometry.

The CRB presented in Equation (13.74) provides a performance bound on source location estimation methods that *jointly* process all the data from all the sensors. Such processing provides the best attainable results, but also requires significant communication bandwidth to transmit data from the individual arrays to the fusion center. Next, we develop approximate performance bounds on schemes that perform bearing estimation at the individual arrays in order to reduce the required communication bandwidth to the fusion center. These CRBs facilitate a study of the tradeoff between source location accuracy and communication bandwidth between the arrays and the fusion center. The methods that we consider are summarized as follows:

1. Each array estimates the source bearing, transmits the bearing estimate to the fusion center, and the fusion processor triangulates the bearings to estimate the source location. This approach does not exploit wavefront coherence between the distributed arrays, but it greatly reduces the communication bandwidth to the fusion center.
2. The raw data from all sensors are jointly processed to estimate the source location. This is the optimum approach that fully utilizes the coherence between distributed arrays, but it requires large communication bandwidth.
3. Combination of methods 1 and 2, where each array estimates the source bearing and transmits the bearing estimate to the fusion center. In addition, the raw data from *one sensor* in each array is transmitted to the fusion center. The fusion center estimates the propagation time delay between pairs of distributed arrays, and processes these time delay estimates with the bearing estimates to localize the source.

© 2005 by Chapman & Hall/CRC

Next we evaluate CRBs for the three schemes for a narrowband source and a wideband source. Consider $H = 3$ identical arrays, each of which contains $N_1 = \cdots = N_H = 7$ sensors. Each array is circular with 4 ft radius, and six sensors are equally spaced around the perimeter and one sensor is in the center. We first evaluate the CRB for a narrowband source with a 1 Hz bandwidth centered at 50 Hz and SNR = 10 dB at each sensor. That is, $G_{s,hh}(\omega)/G_w(\omega) = 10$ for $h = 1, \ldots, H$ and $2\pi(49.5) < \omega lt 2\pi$ (50.5) rad/s. The signal coherence $\gamma_{s,gh}(\omega) = \gamma_s(\omega)$ is varied between 0 and 1. We assume that $T = 4000$ time samples are obtained at each sensor with sampling rate $f_s = 2000$ samples/s. The source localization performance is evaluated by computing the ellipse in (x, y) coordinates that satisfies the expression

$$\begin{bmatrix} x & y \end{bmatrix} \mathbf{J} \begin{bmatrix} x \\ y \end{bmatrix} = 1$$

where $\mathbf{J}$ is the FIM in Equation (13.74). If the errors in (x, y) localization are jointly Gaussian distributed, then the ellipse represents the contour at one standard deviation in root-mean-square (RMS) error. The error ellipse for any unbiased estimator of source location cannot be smaller than this ellipse derived from the FIM.

The $H = 3$ arrays are located at coordinates $(x_1, y_1) = (0, 0)$, $(x_2, y_2) = (400, 400)$, and $(x_3, y_3) = (100, 0)$, where the units are meters. One source is located at $(x_s, y_s) = (200, 300)$, as illustrated in Figure 13.8(a). The RMS error ellipses for joint processing of all sensor data for coherence values $\gamma_s(\omega) = 0$, 0.5, and 1 are also shown in Figure 13.8(a). The coherence between all pairs of arrays is assumed to be identical, i.e. $\gamma_{s,gh}(\omega) = \gamma_s(\omega)$ for $(g, h) = (1, 2), (1, 3), (2, 3)$. The largest ellipse in Figure 13.8(a) corresponds to incoherent signals, i.e. $\gamma_s(\omega) = 0$, and characterizes the performance of the simple method of triangulation using the bearing estimates from the three arrays. Figure 13.8(b) shows the ellipse radius $= \left[(\text{major axis})^2 + (\text{minor axis})^2\right]^{1/2}$ for various values of the signal coherence $\gamma_s(\omega)$. The ellipses for $\gamma_s(\omega) = 0.5$ and 1 are difficult to see in Figure 13.8(a) because they fall on the lines of the $\times$ that marks the source location, illustrating that signal coherence between the arrays significantly improves the CRB on source localization accuracy. Note also that, for this scenario, the localization scheme based on bearing estimation with each array and TDE using one sensor from each array has the same CRB as the optimum, joint processing scheme. Figure 13.8(c) shows a closer view of the error ellipses for the scheme of bearing estimation plus TDE with one sensor from each array. The ellipses are identical to those in Figure 13.8(a) for joint processing.

Figure 13.8 (d)–(f) present corresponding results for a wideband source with bandwidth 20 Hz centered at 50 Hz and SNR 16 dB. That is, $G_{s,hh}/G_w = 40$ for $2\pi(40) < \omega < 2\pi(60)$ rad/s, $h = 1, \ldots, H$. $T = 2000$ time samples are obtained at each sensor with sampling rate $f_s = 2000$ samples/s, so the observation time is 1 s. As in the narrowband case in Figure 13.8 (a)–(c), joint processing reduces the CRB compared with bearings-only triangulation, and bearing plus TDE is nearly optimum.

The CRB provides a lower bound on the variance of unbiased estimates, so an important question is whether an estimator can *achieve* the CRB. We show next in Section 13.3.2.3 that the coherent processing CRBs for the narrowband scenario illustrated in Figure 13.8 (a)–(c) are achievable only when the the coherence is perfect, i.e. $\gamma_s = 1$. Therefore, for that scenario, bearings-only triangulation is optimum in the presence of even small coherence losses. However, for the wideband scenario illustrated in Figure 13.8 (d)–(f), the coherent processing CRBs are achievable for coherence values $\gamma_s > 0.75$.

13.3.2.3 TDE and Examples

The CRB results presented in Section 13.3.2.2 indicate that TDE between widely spaced sensors may be an effective way to improve the source localization accuracy with joint processing. Fundamental performance limits for passive time delay and Doppler estimation have been studied extensively for several decades, e.g. see the collection of papers in Ref. [86]. The fundamental limits are usually parameterized in terms of the SNR at each sensor, the spectral support of the signals (fractional

© 2005 by Chapman & Hall/CRC

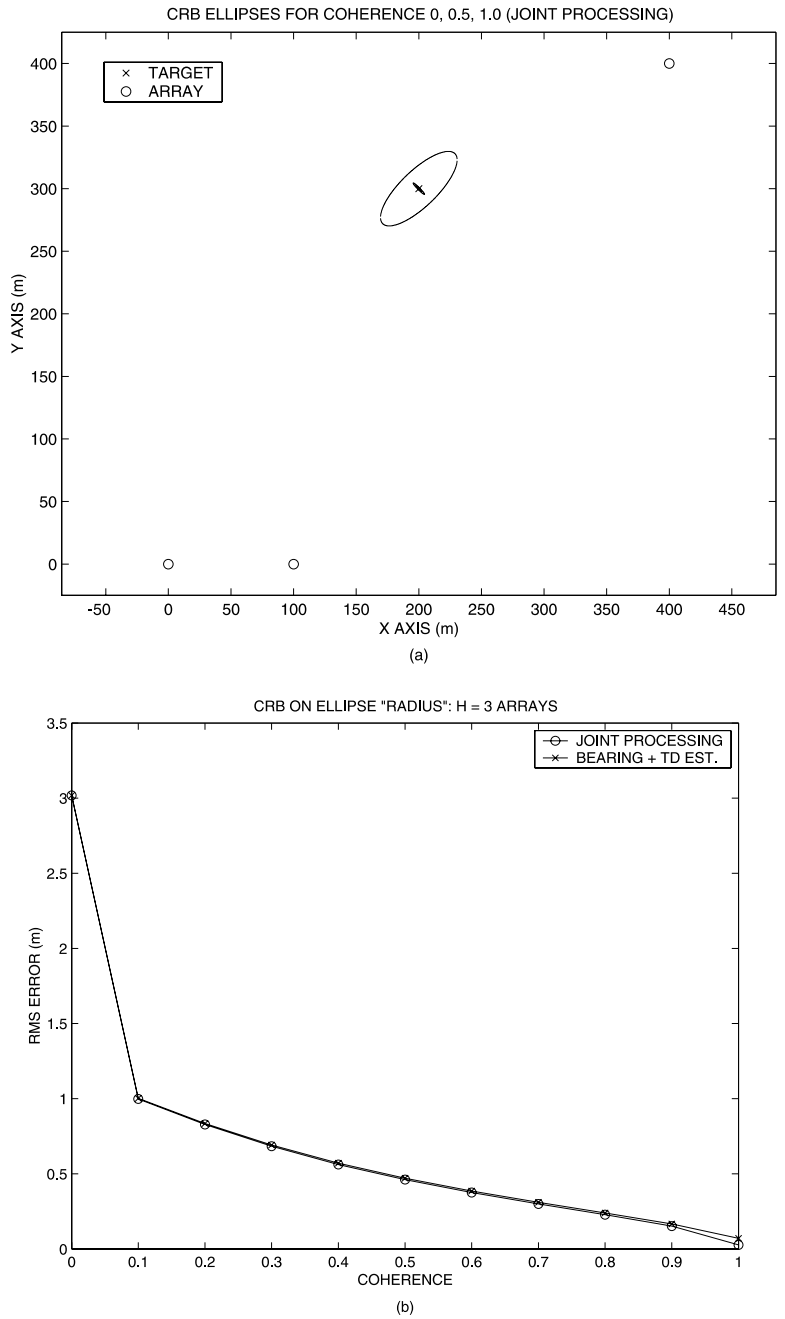
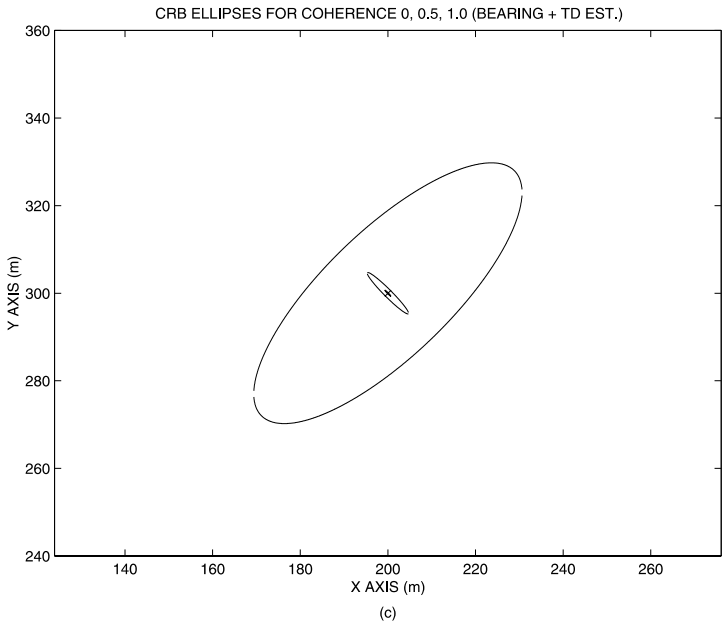


Figure 13.8. RMS source localization error ellipses based on the CRB for $H = 3$ arrays and one *narrowband* source in (a)–(c) and one *wideband* source in (d)–(f). (Originally published in [16], ©2004 IEEE, reprinted with permission.)

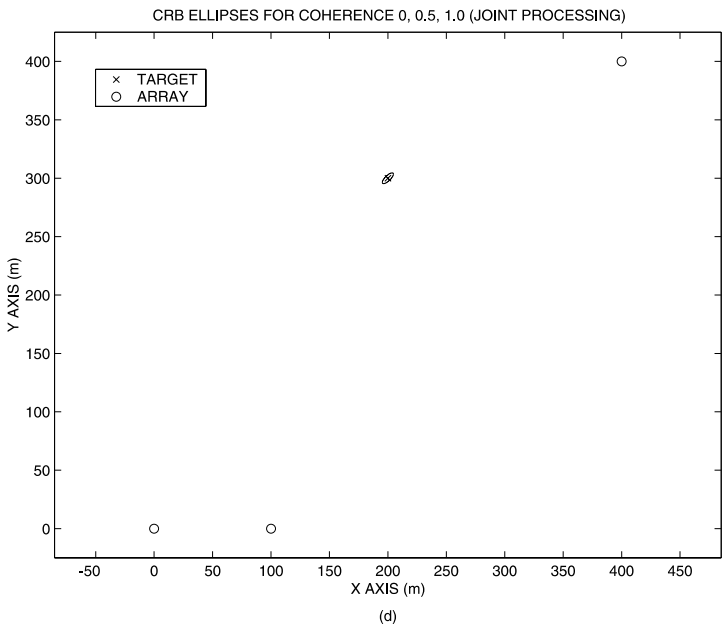
bandwidth), and the time–bandwidth product of the observations. However, the effect of coherence loss on TDE accuracy has not been considered explicitly.

In this section, we quantify the effect of partial signal coherence on TDE. We present Cramér–Rao and Ziv–Zakai bounds that are explicitly parameterized by the signal coherence, along with the

© 2005 by Chapman & Hall/CRC



(c)



(d)

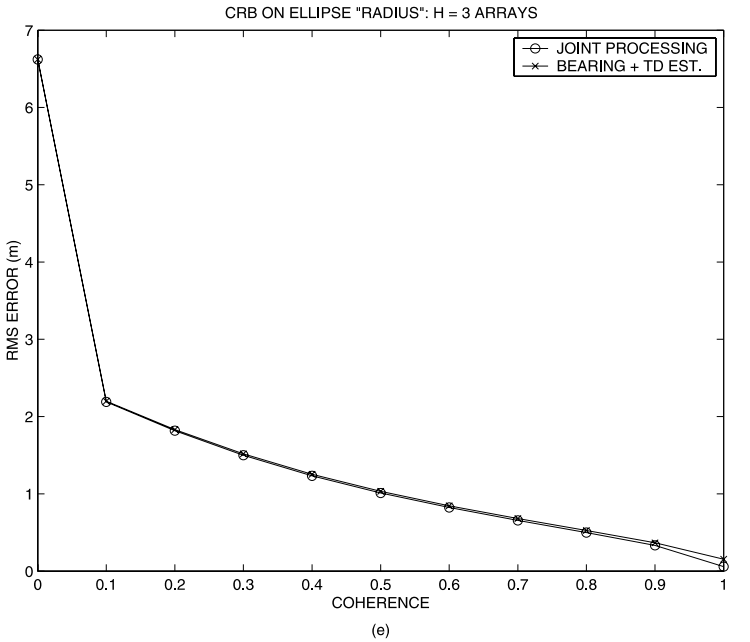
Figure 13.8. Continued.

traditional parameters of SNR, fractional bandwidth, and time–bandwidth product. This analysis of TDE is relevant to method 3 in Section 13.3.2.2. We focus on the case of $H = 2$ sensors here. The extension to $H > 2$ sensors is outlined in Ref. [16].

Let us specialize Equation (13.68) to the case of two sensors, with $H = 2$ and $N_1 = N_2 = 1$, so

$$z_1(t) = s_1(t) + w_1(t) \quad \text{and} \quad z_2(t) = s_2(t - D) + w_2(t) \tag{13.75}$$

© 2005 by Chapman & Hall/CRC



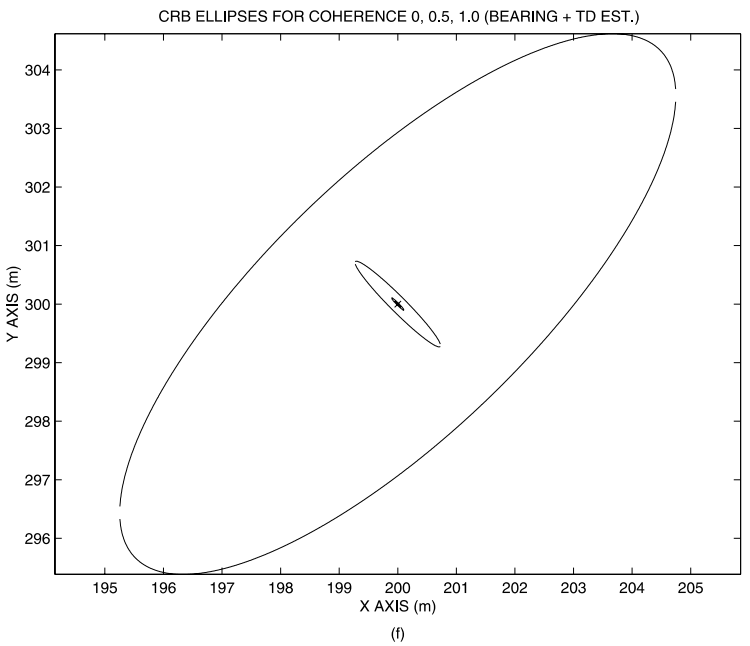


Figure 13.8. Continued.

where $D = D_{21}$ is the differential time delay. Following (73), the CSD matrix is

$$\mathrm{CSD}\begin{bmatrix} z_1(t) \\ z_2(t) \end{bmatrix} = \mathbf{G}_{\mathbf{Z}}(\omega) = \begin{bmatrix} G_{s,11}(\omega) + G_w(\omega) & \mathrm{e}^{+\mathrm{j}\omega D}\gamma_{s,12}(\omega)\left[G_{s,11}(\omega)G_{s,22}(\omega)\right]^{1/2} \\ \mathrm{e}^{-\mathrm{j}\omega D}\gamma_{s,12}(\omega)^*\left[G_{s,11}(\omega)G_{s,22}(\omega)\right]^{1/2} & G_{s,22}(\omega) + G_w(\omega) \end{bmatrix} \tag{13.76}$$

© 2005 by Chapman & Hall/CRC

The signal coherence function $\gamma_{s,12}(\omega)$ describes the degree of correlation that remains in the signal emitted by the source at each frequency ω after propagating to sensors 1 and 2.

We consider the following simplified scenario. The signal and noise spectra are flat over a bandwidth of $\Delta\omega$ rad/s centered at ω_0 rad/s, the observation time is $\mathcal{T}$ seconds, and the propagation is fully saturated, so the signal mean is zero. Further, the signal PSDs are identical at each sensor, and we define the following constants for notational simplicity:

$$G_{s,11}(\omega_0) = G_{s,22}(\omega_0) = G_s, \quad G_w(\omega_0) = G_w, \quad \gamma_{s,12}(\omega_0) = \gamma_s \tag{13.77}$$

Then we can use Equation (13.76) in Equation (13.74) to find the CRB for TDE with $H = 2$ sensors, yielding

$$\mathrm{CRB}(D) = \frac{1}{2\omega_0^2\,(\Delta\omega\mathcal{T}/2\pi)\left[1 + (1/12)(\Delta\omega/\omega_0)^2\right]}\left[\frac{1}{|\gamma_s|^2}\left(1 + \frac{1}{(G_s/G_w)}\right)^2 - 1\right] \tag{13.78}$$

$$> \frac{1}{2\omega_0^2(\Delta\omega\,\mathcal{T}/2\pi)\left[1 + (1/12)(\Delta\omega/\omega_0)^2\right]}\left[\frac{1}{|\gamma_s|^2} - 1\right] \tag{13.79}$$

The quantity $(\Delta\omega\mathcal{T}/2\pi)$ is the time–bandwidth product of the observations, $(\Delta\omega/\omega_0)$ is the fractional bandwidth of the signal, and G_s/G_w is the SNR at each sensor. Note from the high-SNR limit in Equation (13.79) that when the signals are partially coherent, so that $|\gamma_s| < 1$, increased source power does not reduce the CRB. Improved TDE accuracy is obtained with partially coherent signals by increasing the observation time $\mathcal{T}$ or changing the spectral support of the signal, which is $[\omega_0 - \Delta\omega/2,\ \omega_0 + \Delta\omega/2]$. The spectral support of the signal is not controllable in passive TDE applications, so increased observation time is the only means for improving the TDE accuracy with partially coherent signals. Source motion becomes more important during long observation times, as we discuss in Section 13.3.3.

We have shown [16] that the CRB on TDE is achievable only when the coherence γ_s exceeds a threshold. The analysis is based on Ziv–Zakai bounds, as in [72,73], and the result is that the coherence must satisfy the following inequality in order for the CRB on TDE in Equation (13.78) to be achievable:

$$|\gamma_s|^2 \geq \frac{(1 + (1/(G_s/G_w)))^2}{1 + (1/\mathrm{SNR}_{\mathrm{thresh}})}, \quad \text{so } |\gamma_s|^2 \geq \frac{1}{1 + (1/\mathrm{SNR}_{\mathrm{thresh}})} \quad \text{as} \quad \frac{G_s}{G_w} \to \infty \tag{13.80}$$

The quantity $\mathrm{SNR}_{\mathrm{thresh}}$ is

$$\mathrm{SNR}_{\mathrm{thresh}} = \frac{6}{\pi^2(\Delta\omega\mathcal{T}/2\pi)}\left(\frac{\omega_0}{\Delta\omega}\right)^2\left\{\varphi^{-1}\left[\frac{1}{24}\left(\frac{\Delta\omega}{\omega_0}\right)^2\right]\right\}^2 \tag{13.81}$$

where $\varphi(y) = 1/\sqrt{2\pi}\int_y^\infty \exp(-t^2/2)\,\mathrm{d}t$. Since $|\gamma_s|^2 \leq 1$, Equation (13.80) is useful only if $G_s/G_w > \mathrm{SNR}_{\mathrm{thresh}}$. Note that the threshold coherence value in Equation (13.80) is a function of the time–bandwidth product $(\Delta\omega\mathcal{T}/2\pi)$, and the fractional bandwidth $(\Delta\omega/\omega_0)$ through the formula for $\mathrm{SNR}_{\mathrm{thresh}}$ in Equation (13.81).

Figure 13.9(a) contains a plot of Equation (13.80) for a particular case in which the signals are in a band centered at $\omega_0 = 2\pi \times 50$ rad/s and the time duration is $\mathcal{T} = 2$ s. Figure 13.9(a) shows the variation in threshold coherence as a function of signal bandwidth $\Delta\omega$. Note that nearly perfect coherence is required when the signal bandwidth is less than 5 Hz (or 10% fractional bandwidth). The threshold coherence drops sharply for values of signal bandwidth greater than 10 Hz (20% fractional

© 2005 by Chapman & Hall/CRC

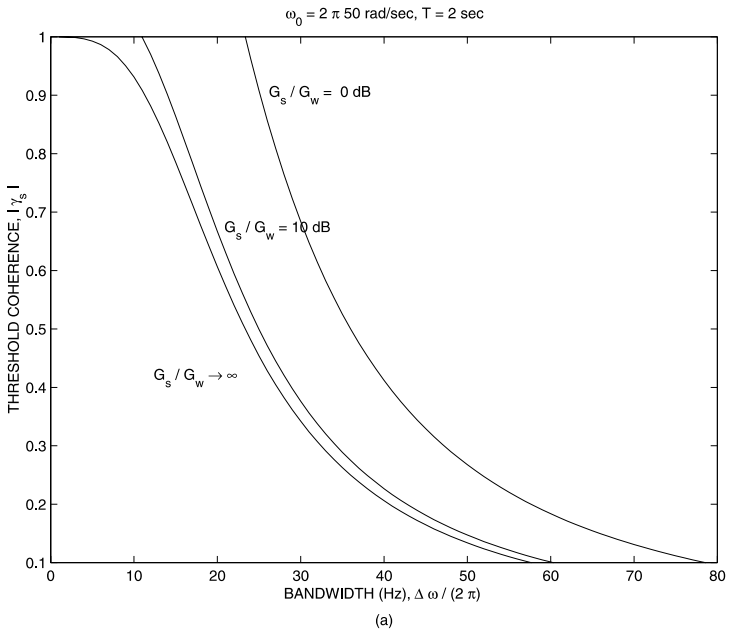


Figure 13.9. Threshold coherence versus bandwidth based on Equation (13.80) for (a) $\omega_0 = 2\pi \times 50$ rad/s, $\mathcal{T} = 2$ s and (b) $\omega_0 = 2\pi \times 100$ rad/s, $\mathcal{T} = 1$ s for SNRs $G_s/G_w = 0$, 10, and ∞ dB. (c) Threshold coherence value from Equation (13.80) versus time–bandwidth product ($\Delta\omega\mathcal{T}/2\pi$) for several values of fractional bandwidth ($\Delta\omega/\omega_0$) and high SNR, $G_s/G_w \to \infty$. (Originally published in [16], ©2004 IEEE, reprinted with permission.)

bandwidth). Thus, for sufficiently wideband signals, e.g. $\Delta\omega \geq 2\pi \times 10$ rad/s, a certain amount of coherence loss can be tolerated while still allowing unambiguous TDE. Figure 13.9(b) shows corresponding results for a case with twice the center frequency and half the observation time. Figure 13.9(c) shows the threshold coherence as a function of the time–bandwidth product and the

© 2005 by Chapman & Hall/CRC

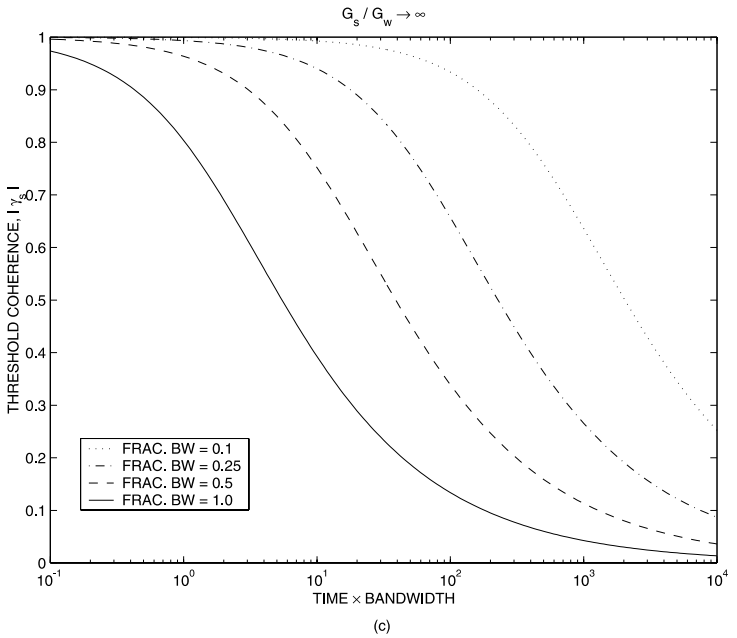


Figure 13.9. Continued.

fractional bandwidth for large SNR, $G_s/G_w \to \infty$. Note that very large time–bandwidth product is required to overcome coherence loss when the fractional bandwidth is small. For example, if the fractional bandwidth is 0.1, then the time–bandwidth product must exceed 100 if the coherence is 0.9. For threshold coherence values in the range from about 0.1 to 0.9, each doubling of the fractional bandwidth reduces the required time–bandwidth product by a factor of 10.

Let us examine a scenario that is typical in aeroacoustics, with center frequency $f_o = \omega_o/(2\pi) = 50\,\text{Hz}$ and bandwidth $\Delta f = \Delta\omega/(2\pi) = 5\,\text{Hz}$, so the fractional bandwidth is $\Delta f/f_o = 0.1$. From Figure 13.9(c), signal coherence $|\gamma_s| = 0.8$ requires time–bandwidth product $\Delta f\,\mathcal{T} > 200$, so the necessary time duration $\mathcal{T} = 40\,\text{s}$ for TDE is impractical for moving sources.

Larger time–bandwidth products of the observed signals are required in order to make TDE feasible in environments with signal coherence loss. As discussed previously, only the observation time is controllable in passive applications, thus leading us to consider source motion models in Section 13.3.3 for use during long observation intervals.

We can evaluate the threshold coherence for the narrowband and wideband scenarios considered in Section 13.3.2.2 for the CRB examples in Figure 13.8. The results are as follows, using Equations (13.80) and (13.81):

1. Narrowband case. $G_s/G_w = 10$, $\omega_0 = 2\pi \times 50\,\text{rad/s}$, $\Delta\omega = 2\pi\,\text{rad/s}$, $\mathcal{T} = 2\,\text{s} \Longrightarrow$ Threshold coherence ≈ 1.
2. Wideband case. $G_s/G_w = 40$, $\omega_0 = 2\pi \times 50\,\text{rad/s}$, $\Delta\omega = 2\pi \times 20\,\text{rad/s}$, $\mathcal{T} = 1\,\text{s} \Longrightarrow$ Threshold coherence ≈ 0.75.

Therefore, for the narrowband case, joint processing of the data from different arrays will not achieve the CRBs in Figure 13.8 (a)–(c) when there is any loss in signal coherence. For the wideband case, joint processing can achieve the CRBs in Figure 13.8 (d)–(f) for coherence values ≥ 0.75.

We have presented simulation examples in [16] that confirm the accuracy of the CRB in Equation (13.78) and threshold coherence in Equation (13.80). In particular, the simulations show that TDE based on cross-correlation processing achieves the CRB only when the threshold coherence is exceeded.

© 2005 by Chapman & Hall/CRC

We conclude this section with a TDE example based on data that were measured by BAE Systems using a synthetically generated, nonmoving, wideband acoustic source. The source bandwidth is approximately 50 Hz with center frequency 100 Hz, so the fractional bandwidth is 0.5. Four nodes are labeled and placed in the locations shown in Figure 13.10(a). The nodes are arranged in a triangle, with nodes on opposite vertices separated by about 330 ft, and adjacent vertices separated by about 230 ft. The source is at node 0, and receiving sensors are located at nodes 1, 2, and 3.

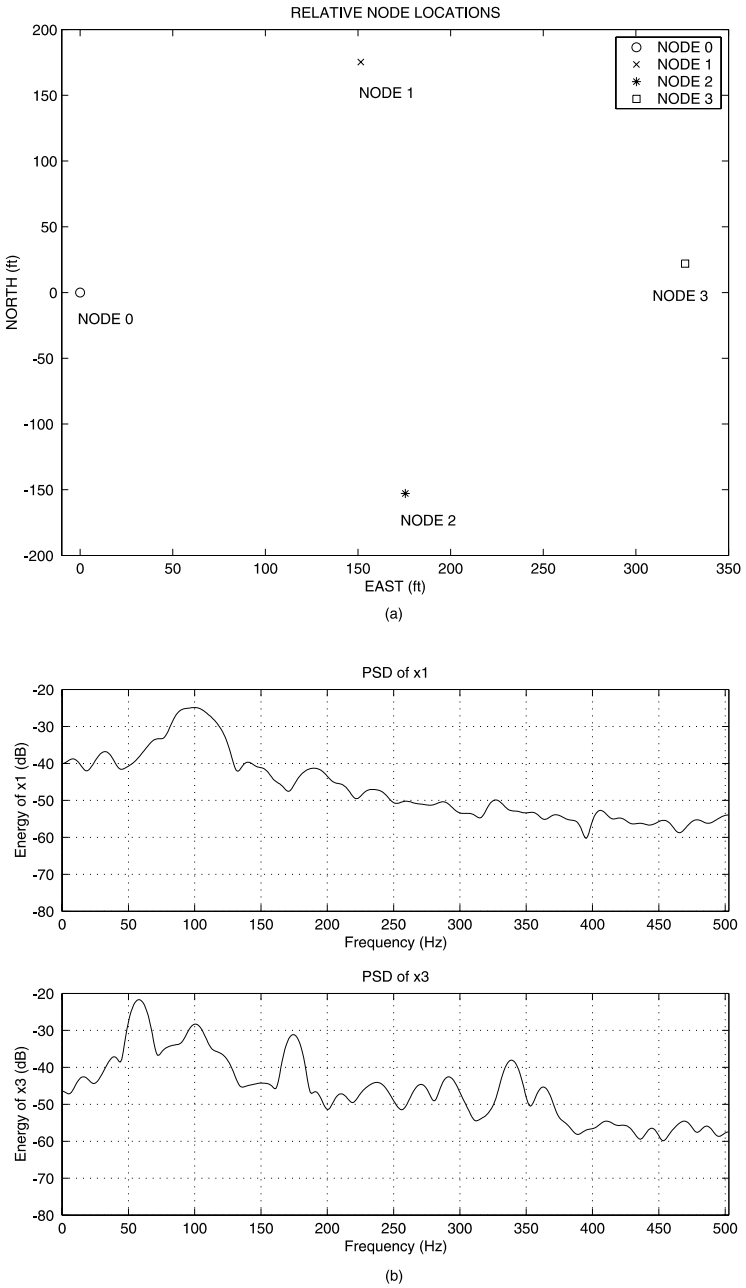


Figure 13.10. (a) Location of nodes. (b) PSDs at nodes 1 and 3 when transmitter is at node 0. (c) Coherence between nodes 1 and 3. (d) Intersection of hyperbolas obtained from differential time delays estimated at nodes 1, 2, and 3. (e) Expanded view of part (d). (Originally published in [16], ©2004 IEEE, reprinted with permission.)

© 2005 by Chapman & Hall/CRC

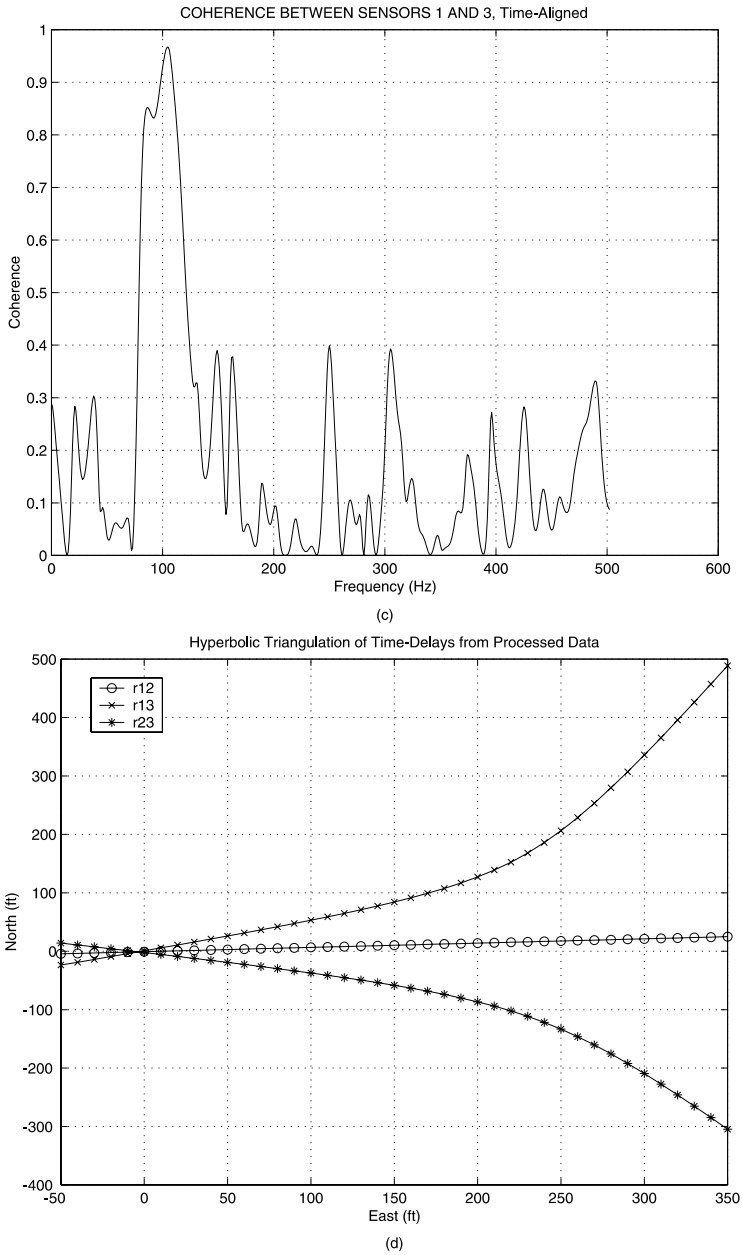


Figure 13.10. Continued.

The PSDs estimated at sensors 1 and 3 are shown in Figure 13.10(b), and the estimated coherence magnitude between sensors 1 and 3 is shown in Figure 13.10(c). The PSDs and coherence are estimated using data segments of duration 1 s. Note that the PSDs are not identical due to differences in the propagation paths. The coherence magnitude exceeds 0.8 over an appreciable band centered at 100 Hz. The threshold coherence value from Equation (13.80) for the parameters in this experiment is 0.5, so the actual coherence of 0.8 exceeds the threshold. Thus, an accurate TDE should be feasible; indeed, we found that generalized cross-correlation yielded accurated TDEs. Differential time delays were estimated using the signals measured at nodes 1, 2, and 3, and the TDEs were hyperbolically triangulated to estimate the location of the source (which is at node 0). Figure 13.10(d) shows the

© 2005 by Chapman & Hall/CRC

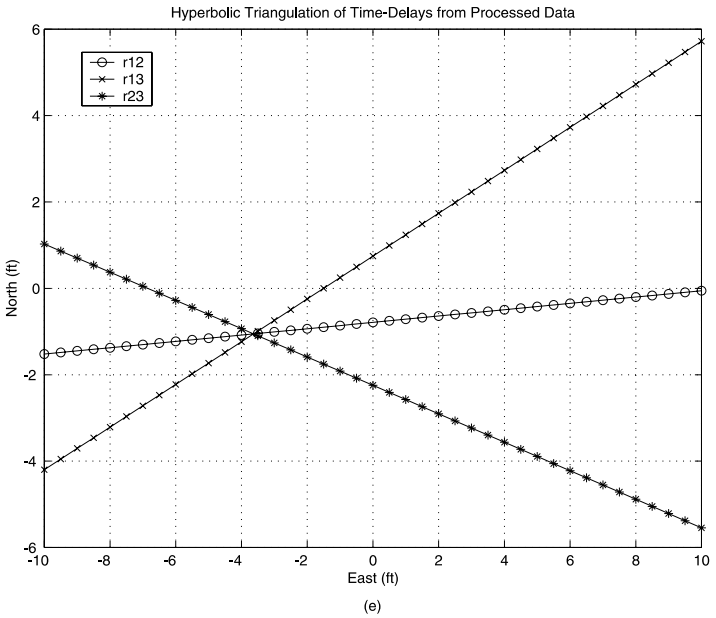


Figure 13.10. Continued.

hyperbolas obtained from the three differential TDE, and Figure 13.10(e) shows an expanded view near the intersection point. The triangulated location is within 1 ft of the true source location, which is at $(-3, 0)$ ft.

This example shows the feasibility of TDE with acoustic signals measured at widely separated sensors, provided that the SNR, fractional bandwidth, time–bandwidth product, and coherence meet the required thresholds. If the signal properties do not satisfy the thresholds, then accurate TDE is not feasible and triangulation of AOAs is optimum.

13.3.3 Tracking Moving Sources

In this section we summarize past work and key issues for tracking moving sources. A widely studied approach for estimating the locations of moving sources with an array of arrays involves bearing estimation at the individual arrays, communication of the bearings to the fusion center, and processing of the bearing estimates at the fusion center with a tracking algorithm (e.g. see Refs [67–71]).

As discussed in Section 13.3.2, jointly processing data from widely spaced sensors has the potential for improved source localization accuracy, compared with incoherent triangulation/tracking of bearing estimates. The potential for improved accuracy depends directly on the TDE between the sensors, which is feasible only with an increased time–bandwidth product of the sensor signals. This leads to a constraint on the minimum observation time $\mathcal{T}$ in passive applications where the signal bandwidth is fixed. If the source is moving, then approximating it as nonmoving becomes poorer as $\mathcal{T}$ increases; so, modeling the source motion becomes more important.

Approximate bounds are known [87,88] that specify the maximum time interval over which moving sources may be approximated as nonmoving for TDE. We have applied the bounds to a typical scenario in aeroacoustics [89]. Let us consider $H = 2$ sensors, and a vehicle moving at 15 m/s (about 5% the speed of sound), with radial motion that is in opposite directions at the two sensors. If the highest frequency of interest is 100 Hz, then the time interval over which the source is well approximated as nonmoving is $\mathcal{T} \ll 0.1$ s. According to the TDE analysis in Section 13.3.2, this yields insufficient time–bandwidth product for partially coherent signals that are typically encountered. Thus, motion modeling

© 2005 by Chapman & Hall/CRC

and Doppler estimation/compensation are critical, even for aeroacoustic sources that move more slowly than in this example.

We have extended the model for a nonmoving source presented in Section 13.3.2 to a moving source with a first-order motion model [89]. We have also presented an algorithm for estimating the motion parameters for multiple moving sources [89], and the algorithm is tested with measured aeroacoustic data. The algorithm is initialized using the local polynomial approximation (LPA) beamformer [90] at each array to estimate the bearings and bearing rates. If the signals have sufficient coherence and bandwidth at the arrays, then the differential TDEs and Doppler shifts may be estimated. The ML solution involves a wideband ambiguity function search over Doppler and TDE [87], but computationally simpler alternatives have been investigated [91]. If TDE is not feasible, then the source may be localized by triangulating bearing, bearing rate, and differential Doppler. Interestingly, differential Doppler provides sufficient information for source localization, even without TDE, as long as five or more sensors are available [92]. Thus, the source motion may be exploited via Doppler estimation in scenarios where TDE is not feasible, such as narrowband or harmonic signals.

Recent work on tracking multiple sources with aeroacoustic sensors includes the penalized ML approach [75] and the α–β/Kalman tracking algorithms [94]. It may be feasible to use source aspect angle differences and Doppler estimation to help solve the data association problem in multiple target tracking based on data from multiple sensor arrays.

13.3.4 Detection and Classification

It is necessary to detect the presence of a source before carrying out the localization processing discussed in Sections 13.3.1, 13.3.2, and 13.3.3. Detection is typically performed by comparing the energy at a sensor with a threshold. The acoustic propagation model presented in Section 13.2 implies that the energy fluctuates due to scattering, so the scattering has a significant impact on detection algorithms and their performance.

In addition to detecting a source and localizing its position, it is desirable to identify (or classify) the type of vehicle from its acoustic signature. The objective is to classify broadly into categories such as "ground, tracked," "ground, wheeled," "airborne, fixed wing," "airborne, rotary wing," and to further identify the particular vehicle type within these categories. Most classification algorithms that have been developed for this problem use the relative amplitudes of harmonic components in the acoustic signal as features to distinguish between vehicle types [95–102]. However, the harmonic amplitudes for a given source may vary significantly due to several factors. The scattering model presented in Section 13.2 implies that the energy in each harmonic will randomly fluctuate due to scattering, and the fluctuations will be stronger at higher frequencies. The harmonic amplitudes may also vary with engine speed and the orientation of the source with respect to the sensor (aspect angle).

In this section, we specialize the scattering model from Section 13.2 to describe the probability distribution for the energy at a single sensor for a source with a harmonic spectrum. We then discuss the implications for detection and classification performance. More detailed discussions may be found in [25] for detection and [93] for classification.

The source spectrum is assumed to be harmonic, with energy at frequencies $\omega_1, \ldots, \omega_L$. Following the notation in Section 13.2.5 and specializing to the case of one source and one sensor, $S(\omega_l)$, $\Omega(\omega_l)$, and $\sigma^2_{\tilde{w}}(\omega_l)$ represent the average source power, the saturation, and the average noise power at frequency ω_l respectively. The complex envelope samples at each frequency ω_l are then modeled with the first element of the vector in Equation (13.55) with $K = 1$ source, and they have a complex Gaussian distribution:

$$\tilde{z}(iT_s; \omega_l) \sim \mathrm{CN}\left(\sqrt{[1-\Omega(\omega_l)]S(\omega_l)}\, e^{j\theta(i;\,\omega_l)}, \quad \Omega(\omega_l)S(\omega_l) + \sigma^2_{\tilde{w}}(\omega_l)\right), \quad \begin{matrix} i = 1, \ldots, T \\ l = 1, \ldots, L \end{matrix} \tag{13.82}$$

The number of samples is T, and the phase $\theta(i;\ \omega_l)$ is defined in Equation (13.21) and depends on the source phase and distance. We allow $\theta(i;\ \omega_l)$ to vary with the time sample index i in case the source

© 2005 by Chapman & Hall/CRC

phase χ or the source distance d_o changes. As discussed in Section 13.2.5, we model the complex Gaussian random variables in Equation (13.82) as independent.

As discussed in Sections 13.2.3 and 13.2.4, the saturation Ω is related to the extinction coefficient of the first moment μ according to $\Omega(\omega_l) = 1 - \exp(-2\,\mu(\omega_l)\,d_o)$, where d_o is the distance from the source to the sensor. The dependence of the saturation on frequency and weather conditions is modeled by the following approximate formula for μ:

$$\mu(\omega) \approx \begin{cases} 4.03 \times 10^{-7} \left(\dfrac{\omega}{2\pi}\right)^2, & \text{mostly sunny} \\ 1.42 \times 10^{-7} \left(\dfrac{\omega}{2\pi}\right)^2, & \text{mostly cloudy} \end{cases} \qquad \frac{\omega}{2\pi} \in [30, 200] \text{ Hz} \tag{13.83}$$

which is obtained by fitting Equation (13.50) to the values for μ^{-1} in Table 13.1. A contour plot of the saturation as a function of frequency and source range is shown in Figure 13.11(a) using Equation (13.83) for mostly sunny conditions. Note that the saturation varies significantly with frequency for ranges > 100 m. Larger saturation values imply more scattering, so the energy in the higher harmonics will fluctuate more widely than the lower harmonics.

We will let $P(\omega_1), \ldots, P(\omega_L)$ denote the estimated energy at each frequency. The energy may be estimated from the complex envelope samples in Equation (13.82) by coherent or incoherent combining:

$$P_{\mathrm{C}}(\omega_l) = \left| \frac{1}{T} \sum_{i=1}^{T} \widetilde{z}(iT_s;\, \omega_l) \mathrm{e}^{-\mathrm{j}\theta(i;\, \omega_l)} \right|^2 \qquad l = 1, \ldots, L \tag{13.84}$$

$$P_{\mathrm{I}}(\omega_l) = \frac{1}{T} \sum_{i=1}^{T} \left| \widetilde{z}(iT_s;\, \omega_l) \right|^2 \qquad l = 1, \ldots, L \tag{13.85}$$

Coherent combining is feasible only if the phase shifts $\theta(i; \omega_l)$ are known or are constant with i. Our assumptions imply that the random variables in Equations (13.84) are independent over l, as are the random variables in Equation (13.85). The probability distribution functions (pdfs) for P_{C} and P_{I} are noncentral chi-squared distributions.[3] We let $\chi^2(D, \delta)$ denote the *standard* noncentral chi-squared distribution with D degrees of freedom and noncentrality parameter δ. Then the random variables in Equations (13.84) and (13.85) may be scaled so that their pdfs are standard noncentral chi-squared distributions:

$$\frac{P_{\mathrm{C}}(\omega_l)}{[\Omega(\omega_l)S(\omega_l) + \sigma_{\tilde{w}}^2(\omega_l)]/2T} \sim \chi^2(2, \delta(\omega_l)) \tag{13.86}$$

$$\frac{P_{\mathrm{I}}(\omega_l)}{[\Omega(\omega_l)S(\omega_l) + \sigma_{\tilde{w}}^2(\omega_l)]/2T} \sim \chi^2(2T, \delta(\omega_l)) \tag{13.87}$$

where the noncentrality parameter is

$$\delta(\omega_l) = \frac{[1 - \Omega(\omega_l)]S(\omega_l)}{\left[\Omega(\omega_l)S(\omega_l) + \sigma_{\tilde{w}}^2(\omega_l)\right]/2T} \tag{13.88}$$

[3]The random variable $\sqrt{P_{\mathrm{C}}}$ in Equation (13.84) has a Rician distribution, which is widely used to model fading RF communication channels.

© 2005 by Chapman & Hall/CRC

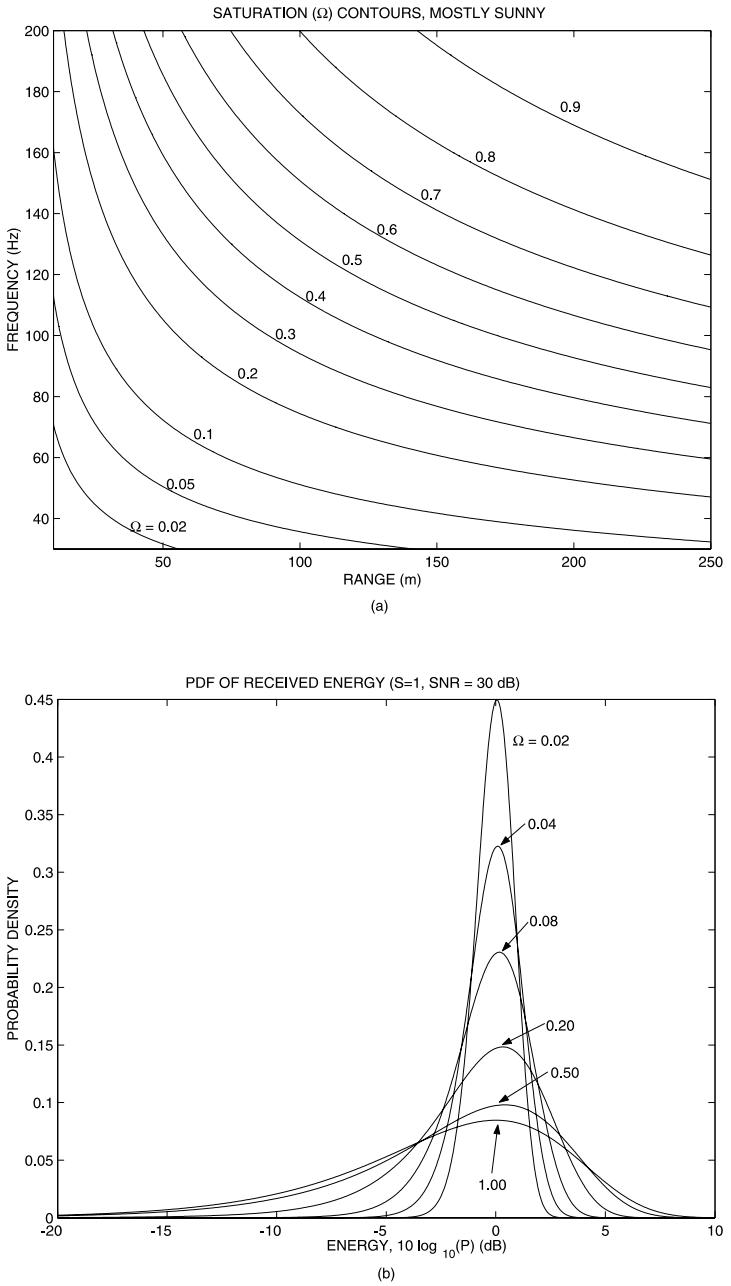


Figure 13.11. (a) Variation of saturation Ω with frequency f and range d_o. (b) Pdf of average power $10\log_{10}(P)$ measured at the sensor for $T = 1$ sample of a signal with $S = 1$ (0 dB), SNR $= 1/\sigma_w^2 = 10^3 = 30$ dB, and various values of the saturation, Ω. (c) Harmonic signature with no scattering. (d) Error bars for harmonic signatures ± one standard deviation caused by scattering at different source ranges.

The only difference in the pdfs for coherent and incoherent combining is the number of degrees of freedom in the noncentral chi-squared pdf: two degrees of freedom for coherent and $2T$ degrees of freedom for incoherent.

The noncentral chi-squared pdf is readily available in analytical form and in statistical software packages, so the performance of detection algorithms may be evaluated as a function of SNR $= S/\sigma_w^2$

© 2005 by Chapman & Hall/CRC

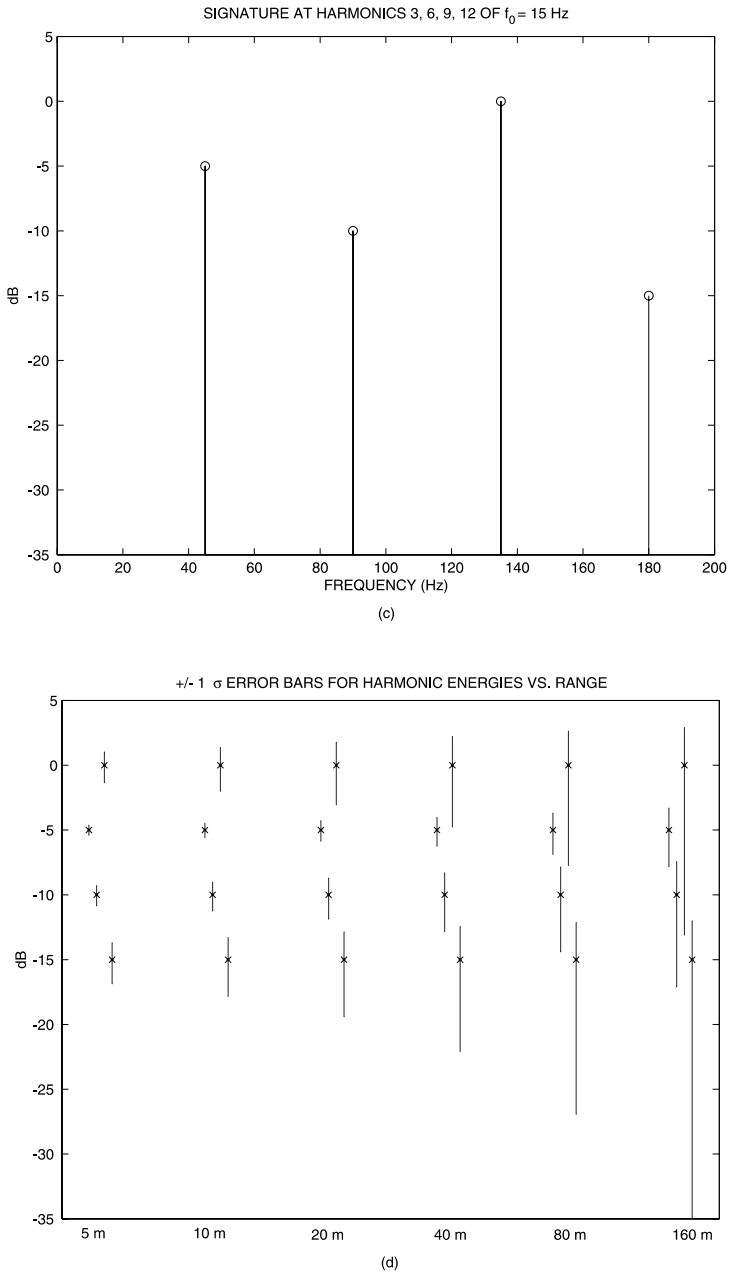


Figure 13.11. Continued.

and saturation Ω. To illustrate the impact of Ω on the energy fluctuations, Figure 13.11(b) shows plots of the pdf of $10\log_{10}(P)$ for $T=1$ sample (so coherent and incoherent are identical), $S=1$, and $\text{SNR} = 1/\sigma_w^2 = 10^3 = 30\,\text{dB}$. Note that a small deviation in the saturation from $\Omega = 0$ causes a significant spread in the distribution of P around the unscattered signal power, $S=1$ (0 dB). This variation in P affects detection performance and limits the performance of classification algorithms that use P as a feature.

Figure 13.12 illustrates signal saturation effects on detection probabilities. In this example, the Neyman–Pearson detection criterion [103] with false-alarm probability of 0.01 was used. The noise is

© 2005 by Chapman & Hall/CRC

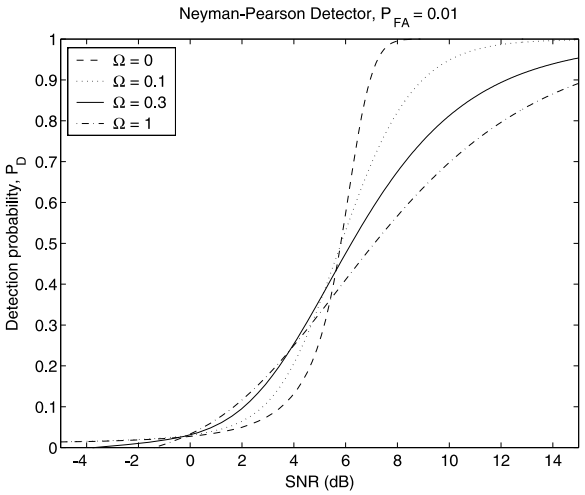


Figure 13.12. Probability of detection as a function of SNR for several values of the saturation parameter Ω. The Neyman–Pearson criterion is used with probability of false alarm $P_{\mathrm{FA}} = 0.01$.

zero-mean Gaussian, as in Section 13.2.2. When $\Omega = 0$, the detection probability is nearly zero for SNR= 2 dB, but it quickly changes to one when the SNR increases by about 6 dB. When $\Omega = 1$, however, the transition is much more gradual: even at SNR= 15 dB, the detection probability is less than 0.9.

The impact of scattering on classification performance can be illustrated by comparing the fluctuations in the measured harmonic signature, $\mathbf{P} = [P(\omega_1), \ldots, P(\omega_L)]^{\mathrm{T}}$, with the "true" signature, $\mathbf{S} = [S(\omega_1), \ldots, S(\omega_L)]^{\mathrm{T}}$, that would be measured in the absence of scattering and additive noise. Figure 13.11(c) and (d) illustrate this variability in the harmonic signature as the range to the target increases. Figure 13.11(c) shows the "ideal" harmonic signature for this example (no scattering and no noise). Figure 13.11(d) shows plus/minus one standard deviation error bars on the harmonics for ranges 5, 10, 20, 40, 80, 160 m under "mostly sunny" conditions, using Equation (13.83). For ranges beyond 80 m, the harmonic components display significant variations, and rank ordering of the harmonic amplitudes would exhibit variations also. The higher frequency harmonics experience larger variations, as expected. Classification based on relative harmonic amplitudes may experience significant performance degradations at these ranges, particularly for sources that have similar harmonic signatures.

13.4 Concluding Remarks

Aeroacoustics has a demonstrated capability for sensor networking applications, providing a low-bandwidth sensing modality that leads to relatively low-cost nodes. In battery-operated conditions, where long lifetime in the field is expected, the node power budget is dominated by the cost of the communications. Consequently, the interplay between the communications and distributed signal processing is critical. We seek optimal network performance while minimizing the communication overhead.

We have considered the impact of the propagation phenomena on our ability to detect, localize, track, and classify acoustic sources. The strengths and limitations of acoustic sensing become clear in this light. Detection ranges and localization accuracy may be reasonably predicted. The turbulent atmosphere introduces spatial coherence losses that impact the ability to exploit large baselines between nodes for increased localization accuracy. The induced statistical fluctuations in amplitude place limits on the ability to classify sources at longer ranges. Very good performance has been demonstrated in

© 2005 by Chapman & Hall/CRC

many experiments; the analysis and experiments described here and elsewhere bound the problem and its solution space.

Because it is passive, and depends on the current atmospheric conditions, acoustic sensing may be strongly degraded in some cases. Passive sensing with high performance in all conditions will very likely require multiple sensing modalities, as well as hierarchical networks. This leads to interesting problems in fusion, sensor density and placement, as well as in distributed processing and communications. For example, when very simple acoustic nodes with the limited capability of measuring loudness are densely deployed, they provide inherent localization capability [104,105]. Such a system, operating at relatively short ranges, provides significant robustness to many of the limitations described here, and may act to queue other sensing modalities for classification or even identification.

Localization based on accurate AOA estimation with short baseline arrays has been carefully analyzed, leading to well-known triangulation strategies. Much more accurate localization, based on cooperative nodes, is possible in some conditions. These conditions depend fundamentally on the time–bandwidth of the observed signal, as well as the spatial coherence. For moving harmonic sources, these conditions are not likely to be supported, whereas sources that are more continuously broadband may be handled in at least some cases. It is important to note that the spatial coherence over a long baseline may be passively estimated in a straightforward way, leading to adaptive approaches that exploit the coherence when it is present. Localization updates, coupled with tracking, lead to an accurate picture of the nonstationary source environment.

Acoustic-based classification is the most challenging signal processing task, due to the source nonstationarities, inherent similarities between the sources, and propagation-induced statistical fluctuations. While the propagation places range limitations on present algorithms, it appears that the source similarities and nonstationarities may be the ultimate limiting factors in acoustic classification. Highly accurate classification will likely require the incorporation of other sensing modalities because of the challenging source characteristics.

Other interesting signal acoustic signal processing includes exploitation of Doppler, hierarchical and multi-modal processing, and handling multipath effects. Complex environments, such as indoor, urban, and forest, create multipaths and diffraction that greatly complicate sensor signal processing and performance modeling. Improved understanding of the impact of these effects, and robust techniques for overcoming them, are needed. Exploitation of the very long-range propagation distances possible with infrasound (frequencies below 20 Hz) [106] also requires further study and experimentation. Finally, we note that strong linkages between the communications network and the sensor signal processing are very important for overall resource utilization, especially including the medium access control (MAC) protocol networking layer.

Acknowledgments

We thank Tien Pham of the Army Research Laboratory for contributions to the wideband AOA estimation material in this chapter, and we thank Sandra Collier of the Army Research Laboratory for many helpful discussions on beamforming in random media.

References

[1] Namorato, M.V., A concise history of acoustics in warfare, *Appl. Acoust.*, 59, 101, 2000.
[2] Becker, G. and Güdesen, A., Passive sensing with acoustics on the battlefield, *Appl. Acoust.*, 59, 149, 2000.
[3] Srour, N. and Robertson, J., Remote netted acoustic detection system, Army Research Laboratory Technical Report, ARL-TR-706, May 1995.
[4] Embleton, T.F.W., Tutorial on sound propagation outdoors, *J. Acoust. Soc. Am.*, 100, 31, 1996.
[5] Tatarskii, V.I., *The Effects of the Turbulent Atmosphere on Wave Propagation,* Keter, Jerusalem, 1971.

© 2005 by Chapman & Hall/CRC

[6] Noble, J.M. *et al.*, The effect of large-scale atmospheric inhomogeneities on acoustic propagation, *J. Acoust. Soc. Am.*, 92, 1040, 1992.
[7] Wilson, D.K. and Thomson, D.W., Acoustic propagation through anisotropic, surface-layer turbulence, *J. Acoust. Soc. Am.*, 96, 1080, 1994.
[8] Norris, D.E. *et al.*, Correlations between acoustic travel-time fluctuations and turbulence in the atmospheric surface layer, *Acta Acust.*, 87, 677, 2001.
[9] Daigle, G.A. *et al.*, Propagation of sound in the presence of gradients and turbulence near the ground, *J. Acoust. Soc. Am.*, 79, 613, 1986.
[10] Ostashev, V.E., *Acoustics in Moving Inhomogeneous Media*, E & FN Spon, London, 1997.
[11] Wilson, D.K., A turbulence spectral model for sound propagation in the atmosphere that incorporates shear and buoyancy forcings, *J. Acoust. Soc. Am.*, 108 (5, Pt. 1), 2021, 2000.
[12] Kay, S.M. *et al.*, Broad-band detection based on two-dimensional mixed autoregressive models, *IEEE Trans. Signal Process.*, 41(7), 2413, 1993.
[13] Agrawal, M. and Prasad, S., DOA estimation of wideband sources using a harmonic source model and uniform linear array, *IEEE Trans. Signal Process.*, 47(3), 619, 1999.
[14] Feder, M., Parameter estimation and extraction of helicopter signals observed with a wide-band interference, *IEEE Trans. Signal Process.*, 41(1), 232, 1993.
[15] Zeytinoglu, M. and Wong, K.M., Detection of harmonic sets, *IEEE Trans. Signal Process.*, 43(11), 2618, 1995.
[16] Kozick, R.J. and Sadler, B.M., Source localization with distributed sensor arrays and partial spatial coherence, *IEEE Trans. Signal Process.*, 52(3), 601–616, 2004.
[17] Morgan, S. and Raspet, R., Investigation of the mechanisms of low-frequency wind noise generation outdoors, *J. Acoust. Soc. Am.*, 92, 1180, 1992.
[18] Bass, H.E. *et al.*, Experimental determination of wind speed and direction using a three microphone array, *J. Acoust. Soc. Am.*, 97, 695, 1995.
[19] Salomons, E.M., *Computational Atmospheric Acoustics*, Kluwer, Dordrecht, 2001.
[20] Kay, S.M., *Fundamentals of Statistical Signal Processing, Estimation Theory*, Prentice-Hall, 1993.
[21] Wilson, D.K., Performance bounds for acoustic direction-of-arrival arrays operating in atmospheric turbulence, *J. Acoust. Soc. Am.*, 103(3), 1306, 1998.
[22] Collier, S.L. and Wilson, D.K., Performance bounds for passive arrays operating in a turbulent medium: plane-wave analysis, *J. Acoust. Soc. Am.*, 113(5), 2704, 2003.
[23] Collier, S.L. and Wilson, D.K., Performance bounds for passive sensor arrays operating in a turbulent medium II: spherical-wave analysis, *J. Acoust. Soc. Am.*, 116(2), 987–1001, 2004.
[24] Ostashev, V.E. and Wilson, D.K., Relative contributions from temperature and wind velocity fluctuations to the statistical moments of a sound field in a turbulent atmosphere, *Acta Acust.*, 86, 260, 2000.
[25] Wilson, D.K. *et al.*, Simulation of detection and beamforming with acoustical ground sensors, *Proceedings of SPIE 2002 AeroSense Symposium*, Orlando, FL, April 1–5, 2002, 50.
[26] Norris, D.E. *et al.*, Atmospheric scattering for varying degrees of saturation and turbulent intermittency, *J. Acoust. Soc. Am.*, 109, 1871, 2001.
[27] Flatté, S.M. *et al.*, *Sound Transmission Through a Fluctuating Ocean*, Cambridge University Press, Cambridge, U.K., 1979.
[28] Daigle, G.A. *et al.*, Line-of-sight propagation through atmospheric turbulence near the ground, *J. Acoust. Soc. Am.*, 74, 1505, 1983.
[29] Bass, H.E. *et al.*, Acoustic propagation through a turbulent atmosphere: experimental characterization, *J. Acoust. Soc. Am.*, 90, 3307, 1991.
[30] Ishimaru, A., *Wave Propagation and Scattering in Random Media*, IEEE Press, New York, 1997.
[31] Havelock, D.I. *et al.*, Measurements of the two-frequency mutual coherence function for sound propagation through a turbulent atmosphere, *J. Acoust. Soc. Am.*, 104(1), 91, 1998.
[32] Paulraj, A. and Kailath, T., Direction of arrival estimation by eigenstructure methods with imperfect spatial coherence of wavefronts, *J. Acoust. Soc. Am.*, 83, 1034, 1988.

© 2005 by Chapman & Hall/CRC

[33] Song, B.-G. and Ritcey, J.A., Angle of arrival estimation of plane waves propagating in random media, *J. Acoust. Soc. Am.*, 99(3), 1370, 1996.
[34] Gershman, A.B. *et al.*, Matrix fitting approach to direction of arrival estimation with imperfect spatial coherence, *IEEE Trans. on Signal Process.*, 45(7), 1894, 1997.
[35] Besson, O. *et al.*, Approximate maximum likelihood estimators for array processing in multiplicative noise environments, *IEEE Trans. Signal Process.*, 48(9), 2506, 2000.
[36] Ringelstein, J. *et al.*, Direction finding in random inhomogeneous media in the presence of multiplicative noise, *IEEE Signal Process. Lett.*, 7(10), 269, 2000.
[37] Stoica, P. *et al.*, Direction-of-arrival estimation of an amplitude-distorted wavefront, *IEEE Trans. Signal Process.*, 49(2), 269, 2001.
[38] Besson, O. *et al.*, Simple and accurate direction of arrival estimator in the case of imperfect spatial coherence, *IEEE Trans. Signal Process.*, 49(4), 730, 2001.
[39] Ghogho, M. *et al.*, Estimation of directions of arrival of multiple scattered sources, *IEEE Trans. Signal Process.*, 49(11), 2467, 2001.
[40] Fuks, G. *et al.*, Bearing estimation in a Ricean channel — Part I: inherent accuracy limitations, *IEEE Trans. Signal Process.*, 49(5), 925, 2001.
[41] Boehme, J.F., Array processing, in *Advances in Spectrum Analysis and Array Processing*, vol. 2, Haykin, S. (ed.), Prentice-Hall, 1991.
[42] Van Trees, H.L., *Optimum Array Processing*, Wiley, 2002.
[43] Owsley, N. Sonar array processing, in *Array Signal Processing,* Haykin, S. (ed.), Prentice-Hall, 1984.
[44] Su, G. and Morf, M., Signal subspace approach for multiple wideband emitter location, *IEEE Trans. Acoust. Speech Signal Process.*, 31(6), 1502, 1983.
[45] Wang, H. and Kaveh, M., Coherent signal-subspace processing for the detection and estimation of angles of arrival of multiple wide-band sources, *IEEE Trans. Acoust. Speech Signal Process.*, ASSP-33(4), 823, 1985.
[46] Swingler, D.N. and Krolik, J., Source location bias in the coherently focused high-resolution broad-band beamformer, *IEEE Trans. Acoust. Speech Signal Process.*, 37(1), 143, 1989.
[47] Valaee, S. and Kabal, P., Wideband array processing using a two-sided correlation transformation, *IEEE Trans. Signal Process.*, 43(1), 160, 1995.
[48] Krolik, J. and Swingler, D., Focused wide-band array processing by spatial resampling, *IEEE Trans. Acoust. Speech Signal Process.*, 38(2), 356, 1990.
[49] Krolik, J., Focused wide-band array processing for spatial spectral estimation, in *Advances in Spectrum Analysis and Array Processing,* Vol. 2, Haykin, S. (ed.), Prentice-Hall, 1991.
[50] Friedlander, B. and Weiss, A.J., Direction finding for wide-band signals using an interpolated array, *IEEE Trans. Signal Process.*, 41(4), 1618, 1993.
[51] Doron, M.A. *et al.*, Coherent wide-band processing for arbitrary array geometry, *IEEE Trans. Signal Process.*, 41(1), 414, 1993.
[52] Buckley, K.M. and Griffiths, L.J., Broad-band signal-subspace spatial-spectrum (BASS-ALE) estimation, *IEEE Trans. Acoust. Speech Signal Process.*, 36(7), 953, 1988.
[53] Agrawal, M. and Prasad, S., Broadband DOA estimation using spatial-only modeling of array data, *IEEE Trans. Signal Process.*, 48(3), 663, 2000.
[54] Sivanand, S. *et al.*, Focusing filters for wide-band direction finding, *IEEE Trans. Signal Process.*, 39(2), 437, 1991.
[55] Sivanand, S. and Kaveh M., Multichannel filtering for wide-band direction finding, *IEEE Trans. Signal Process.*, 39(9), 2128, 1991.
[56] Sivanand, S., On focusing preprocessor for broadband beamforming, in *Sixth SP Workshop on Statistical Signal and Array Processing,* Victoria, BC, Canada, October 1992, 350.
[57] Ward, D.B. *et al.*, Broadband DOA estimation using frequency invariant beamforming, *IEEE Trans. Signal Process.*, 46(5), 1463, 1998.

© 2005 by Chapman & Hall/CRC

[58] Bangs, W.J., Array processing with generalized beamformers, PhD Dissertation, Yale University, 1972.
[59] Swingler, D.N., An approximate expression for the Cramer–Rao bound on DOA estimates of closely spaced sources in broadband line-array beamforming, *IEEE Trans. Signal Process.*, 42(6), 1540, 1994.
[60] Yang, J. and Kaveh, M., Coherent signal-subspace transformation beamformer, *IEE Proc.,* 137 (Pt. F, 4), 267, 1990.
[61] Pham, T. and Sadler, B.M., Acoustic tracking of ground vehicles using ESPRIT, in *SPIE Proc. Volume 2485, Automatic Object Recognition* V, Orlando, FL, April 1995, 268.
[62] Pham, T. *et al.*, High resolution acoustic direction finding algorithm to detect and track ground vehicles, in *20th Army Science Conference,* Norfolk, VA, June 1996; see also *Twentieth Army Science Conference, Award Winning Papers*, World Scientific, 1997.
[63] Pham, T. and Sadler, B.M., Adaptive wideband aeroacoustic array processing, in *8th IEEE Statistical Signal and Array Processing Workshop*, Corfu, Greece, June 1996, 295.
[64] Pham, T. and Sadler, B.M., Adaptive wideband aeroacoustic array processing, in *Proceedings of the 1st Annual Conference of the Sensors and Electron Devices Federated Laboratory Research Program,* College Park, MD, January 1997.
[65] Pham, T. and Sadler, B.M., Focused wideband array processing algorithms for high-resolution direction finding, in *Proceedings of MSS Specialty Group on Acoustics and Seismic Sensing,* September 1998.
[66] Pham, T. and Sadler, B.M., Wideband array processing algorithms for acoustic tracking of ground vehicles, in *Proceedings 21st Army Science Conference,* 1998.
[67] Tenney, R.R. and Delaney, J.R., A distributed aeroacoustic tracking algorithm, in *Proceedings of the American Control Conference*, June 1984, 1440.
[68] Bar-Shalom, Y. and Li, X.-R., *Multitarget-Multisensor Tracking: Principles and Techniques*, YBS, 1995.
[69] Farina, A., Target tracking with bearings-only measurements, *Signal Process.*, 78, 61, 1999.
[70] Ristic, B. *et al.*, The influence of communication bandwidth on target tracking with angle only measurements from two platforms, *Signal Process.*, 81, 1801, 2001.
[71] Kaplan, L.M. *et al.*, Bearings-only target localization for an acoustical unattended ground sensor network, in *Proceedings of SPIE AeroSense*, Orlando, Florida, April 2001.
[72] Weiss, A.J. and Weinstein, E., Fundamental limitations in passive time delay estimation — part 1: narrowband systems, *IEEE Trans. Acoust. Speech Signal Process.*, ASSP-31(2), 472, 1983.
[73] Weinstein, E. and Weiss, A.J., Fundamental limitations in passive time delay estimation — part 2: wideband systems, *IEEE Trans. Acoust. Speech Signal Process.*, ASSP-32(5), 1064, 1984.
[74] Bell, K., Wideband direction of arrival (DOA) estimation for multiple aeroacoustic sources, in *Proceedings of 2000 Meeting of the MSS Specialty Group on Battlefield Acoustics and Seismics*, Laurel, MD, October 18–20, 2000.
[75] Bell, K., Maximum a posteriori (MAP) multitarget tracking for broadband aeroacoustic sources, in *Proceedings of 2001 Meeting of the MSS Specialty Group on Battlefield Acoustics and Seismics*, Laurel, MD, October 23–26, 2001.
[76] Wax, M. and Kailath, T., Decentralized processing in sensor arrays, *IEEE Trans. Acoust. Speech Signal Process.*, ASSP-33(4), 1123, 1985.
[77] Stoica, P. *et al.*, Decentralized array processing using the MODE algorithm, *Circuits, Syst. Signal Process.*, 14(1), 17, 1995.
[78] Weinstein, E., Decentralization of the Gaussian maximum likelihood estimator and its applications to passive array processing, *IEEE Trans. Acoust. Speech Signal Process.*, ASSP-29(5), 945, 1981.
[79] Moses, R.L. and Patterson, R., Self-calibration of sensor networks, in *Proceedings of SPIE AeroSense 2002*, 4743, April 2002, 108.

© 2005 by Chapman & Hall/CRC

[80] Spiesberger, J.L., Locating animals from their sounds and tomography of the atmosphere: experimental demonstration, *J. Acoust. Soc. Am.*, 106, 837, 1999.
[81] Wilson, D.K. *et al.*, An overview of acoustic travel-time tomography in the atmosphere and its potential applications, *Acta Acust.*, 87, 721, 2001.
[82] Ferguson, B.G., Variability in the passive ranging of acoustic sources in air using a wavefront curvature technique, *J. Acoust. Soc. Am.*, 108(4), 1535, 2000.
[83] Ferguson, B.G., Time-delay estimation techniques applied to the acoustic detection of jet aircraft transits, *J. Acoust. Soc. Am.*, 106(1), 255, 1999.
[84] Friedlander, B., On the Cramer–Rao bound for time delay and doppler estimation, *IEEE Trans. Info. Theory*, IT-30(3), 575, 1984.
[85] Whittle, P., The analysis of multiple stationary time series, *J. R. Stat. Soc.*, 15, 125, 1953.
[86] Carter, G.C. (ed.), *Coherence and Time Delay Estimation* (Selected Reprint Volume), IEEE Press, 1993.
[87] Knapp, C.H. and Carter, G.C., Estimation of time delay in the presence of source or receiver motion, *J. Acoust. Soc. Am.*, 61(6), 1545, 1977.
[88] Adams, W.B. *et al.*, Correlator compensation requirements for passive time-delay estimation with moving source or receivers, *IEEE Trans. Acoust. Speech Signal Process.*, ASSP-28(2), 158, 1980.
[89] Kozick, R.J. and Sadler, B.M., Tracking moving acoustic sources with a network of sensors, Army Research Laboratory Technical Report ARL-TR-2750, October 2002.
[90] Katkovnik, V. and Gershman, A.B., A local polynomial approximation based beamforming for source localization and tracking in nonstationary environments, *IEEE Signal Process. Lett.*, 7(1), 3, 2000.
[91] Betz, J.W., Comparison of the deskewed short-time correlator and the maximum likelihood correlator, *IEEE Trans. Acoust. Speech Signal Process.*, ASSP-32(2), 285, 1984.
[92] Schultheiss, P.M. and Weinstein, E., Estimation of differential Doppler shifts, *J. Acoust. Soc. Am.*, 66(5), 1412, 1979.
[93] Kozick, R.J. and Sadler, B.M., Information sharing between localization, tracking, and identification algorithms, in *Proceedings of 2002 Meeting of the MSS Specialty Group on Battlefield Acoustics and Seismics*, Laurel, MD, September 24–27, 2002.
[94] Damarla, T.R. *et al.*, Army acoustic tracking algorithm, in *Proceedings of 2002 Meeting of the MSS Specialty Group on Battlefield Acoustics and Seismics*, Laurel, MD, September 24–27, 2002.
[95] Wellman, M. *et al.*, Acoustic feature extraction for a neural network classifier, Army Research Laboratory, ARL-TR-1166, January 1997.
[96] Srour, N. *et al.*, Utilizing acoustic propagation models for robust battlefield target identification, in *Proceedings of 1998 Meeting of the IRIS Specialty Group on Acoustic and Seismic Sensing*, September 1998.
[97] Lake, D., Robust battlefield acoustic target identification, in *Proceedings of 1998 Meeting of the IRIS Specialty Group on Acoustic and Seismic Sensing*, September 1998.
[98] Lake, D., Efficient maximum likelihood estimation for multiple and coupled harmonics, Army Research Laboratory, ARL-TR-2014, December 1999.
[99] Lake, D., Harmonic phase coupling for battlefield acoustic target identification, in *Proceedings IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2049, 1998.
[100] Hurd, H. and Pham, T., Target association using harmonic frequency tracks, in *Proceedings of Fifth IEEE International Conference on Information Fusion*, 2002, 860.
[101] Wu, H. and Mendel, J.M., Data analysis and feature extraction for ground vehicle identification using acoustic data, in *2001 MSS Specialty Group Meeting on Battlefield Acoustics and Seismic Sensing*, Johns Hopkins University, Laurel, MD, October 2001.
[102] Wu, H. and Mendel, J.M., Classification of ground vehicles from acoustic data using fuzzy logic rule-based classifiers: early results, in *Proceedings of SPIE AeroSense*, Orland, FL, April 1–5, 2002, 62.

© 2005 by Chapman & Hall/CRC

[103] Kay, S.M., *Fundamentals of Statistical Signal Processing, Detection Theory*, Prentice-Hall, 1998.
[104] Pham, T. and Sadler, B.M., Energy-based detection and localization of stochastic signals, in *2002 Meeting of the MSS Specialty Group on Battlefield Acoustic and Seismic Sensing*, Laurel, MD, September 2002.
[105] Pham, T., Localization algorithms for ad-hoc network of disposable sensors, in *2003 MSS National Symposium on Sensor and Data Fusion*, San Diego, CA, June 2003.
[106] Bedard, A.J. and Georges, T.M., Atmospheric infrasound, *Phys. Today*, 53, 32, 2000.

© 2005 by Chapman & Hall/CRC

14

Distributed Multi-Target Detection in Sensor Networks

Xiaoling Wang, Hairong Qi, and Steve Beck

14.1 Introduction

Recent advances in micro-electro-mechanical systems (MEMS), wireless communication technologies, and digital electronics are responsible for the emergence of sensor networks that deploy thousands of low-cost sensor nodes integrating sensing, processing, and communication capabilities. These sensor networks have been employed in a wide variety of applications, ranging from military surveillance to civilian and environmental monitoring. Examples of such applications include battlefield command, control, and communication [1], target detection, localization, tracking and classification [2–6], transportation monitoring [7], pollution monitoring in the air, soil, and water [8,9], ecosystem monitoring [10], etc.

A fundamental problem concerning these different sensor network applications is to detect the targets in the field of interest. This problem has two levels of difficulty: single target detection and multiple target detection. The single target detection problem can be solved using some off-the-shelf methods, e.g. a constant false-alarm rate detector on the acoustic signals can determine presence of a target if the signal energy exceeds an adaptive threshold. On the other hand, the multiple target detection problem is rather challenging and very difficult to solve.

Over the years, researchers have employed different sensing modalities, one-dimensional (1-D) or two-dimensional (2-D), to detect the targets. For example, 2-D imagers are widely used tools. Through image segmentation, the targets of interest can be separated from the background and later identified using pattern classification methods. However, if multiple targets appear overlapped with each other in a single image frame, or the target pixels are mixed with background clutter, which is almost always the case, then detecting these targets from images can be extremely difficult. In such situations, 1-D signals, such as the acoustic and seismic signals, may offer advantages because of the intrinsic correlation among the target signatures, the captured signals from multiple sensor nodes, and their relative positions. For example, the acoustic signal received at an individual sensor node can be regarded as a linear/nonlinear weighted combination of the signals radiated from the targets with the weights determined by the signal propagation model and the distance between the targets and the sensor node.

© 2005 by Chapman & Hall/CRC

The problem of detecting multiple targets in sensor networks from their linear/nonlinear mixtures is similar to the traditional blind source separation (BSS) problem [11,12], where the different targets in the field are considered as the sources. The "blind" qualification of BSS refers to the fact that there is no *a priori* information available on the number of sources, the distribution of sources, or the mixing model [13]. Independent component analysis (ICA) [14–17] has been a widely accepted technique to solve the BSS problem. Although the BSS problem involves two implications, source number estimation and source separation, for conceptual and computational simplicity, most ICA algorithms employ the linear instantaneous mixture model and make the assumption that the number of sources equals the number of observations, so that the mixing/unmixing matrix is square and can be easily estimated. However, this equality assumption is generally not the case in sensor network applications, where thousands of sensors can be densely deployed within the sensing field and the number of sensors can easily exceed the number of sources. Hence, the number of sources has to be estimated before any further calculations can be done.

Despite the active research in ICA source separation algorithms, less attention has been paid to the problem of source number estimation, which is also referred to as the problem of model order estimation [18]. Several approaches have been put forward on this problem so far, some heuristic, others based on more principled approaches [19–21]. As discussed in Ref. [18], in recent years, it has become clear that techniques of the latter category are superior and, at best, heuristic methods may be seen as approximations to more detailed underlying principles.

Most model order estimation methods developed to date require centralized processing and are derived under the assumption that a long observed sequence from all the involved sensors is available in order to estimate the most probable number of sources and the mixing matrix. However, this assumption is not appropriate for real-time processing in sensor networks for both the sheer amount of sensor nodes deployed in the field and the limited power supply on the battery-supported sensor nodes.

In this chapter, a distributed multiple target detection framework is developed for sensor network applications based on the centralized blind source estimation techniques. The outline of this chapter is as follows. We first describe the problem of BSS separation in Section 14.2 and source number estimation in Section 14.3. Based on the background introduction of these two related problems, we then present a distributed source number estimation technique for multiple target detection in sensor networks. We also conduct experiments to evaluate the performance of the proposed distributed method compared with the existing centralized approach.

14.2 The BSS Problem

The BSS problem [11,12] considers how to extract source signals from their linear or nonlinear mixtures using a minimum of *a priori* information. The most intuitive example of the BSS problem is the so-called *cocktail-party problem* [15]. Suppose there are two people speaking simultaneously in a room and two microphones placed in different locations of the room. Let $x_1(t)$ and $x_2(t)$ denote the amplitude of the speech signals *recorded* at the two microphones, and let $s_1(t)$ and $s_2(t)$ be the amplitude of the speech signals *generated* by the two speakers. We call $x_1(t)$ and $x_2(t)$ the *observed signals* and $s_1(t)$ and $s_2(t)$ the *source signals*. Intuitively, we know that both the observed signals are mixtures of the two source signals. If we assume that the mixing process is linear, then we can model it using Equation (14.1), where the observed signals ($x_1(t)$ and $x_2(t)$) are weighted sums of the source signals ($s_1(t)$ and $s_2(t)$), and a_{11}, a_{12}, a_{21}, and a_{22} denote the weights, which are normally dependent upon the distances between the microphones and the speakers.

$$\begin{aligned} x_1(t) &= a_{11}s_1(t) + a_{12}s_2(t) \\ x_2(t) &= a_{21}s_1(t) + a_{22}s_2(t) \end{aligned} \tag{14.1}$$

In many circumstances, it is desired to estimate the source signals from the observed signals only in order to identify the sources. If the a_{ij} values are known, then the solutions to the linear equations in

© 2005 by Chapman & Hall/CRC

Equation (14.1) are straightforward. However, this is not always the case; if the a_{ij} values are unknown, then the problem is considerably more difficult. A common approach is to adopt some statistical properties of the source signals to help estimate the weights a_{ij}. For example, the ICA algorithms are developed on the assumption that the source signals $s_i(t)$, at each time instant t, are statistically independent.

In sensor networks, sensor nodes are usually densely deployed in the field. For the multiple target detection problem, if the targets are close to each other, then the observation from each individual sensor node is also a mixture of the source signals generated by the targets. Therefore, the basic formulation of the BSS problem and its ICA-based solution are applicable to the problem of multiple target detection in sensor networks.

Suppose there are m targets in the sensor field generating the source signals $s_i(t)$, $i = 1, \ldots, m$ and n sensor observations recorded at the sensor nodes $x_j(t)$, $j = 1, \ldots, n$, where $t = 1, \ldots, T$ indicates the time index of the discrete-time signals and we use p to represent the number of discrete times. Then the sources and the observed mixtures at t can be denoted as vectors $\mathbf{s}(t) = [s_1(t), \ldots, s_m(t)]^{\mathrm{T}}$ and $\mathbf{x}(t) = [x_1(t), \ldots, x_n(t)]^{\mathrm{T}}$ respectively. Let $\mathbf{X}_{n\times p} = \{\mathbf{x}(t)\}$ represent the sensor observation matrix, $\mathbf{S}_{m\times p} = \{\mathbf{s}(t)\}$ the unknown source matrix, and assume the mixing process is linear; then, $\mathbf{X}$ can be represented as

$$\mathbf{X} = \mathbf{AS} \tag{14.2}$$

where $\mathbf{A}_{n\times m}$ is the unknown nonsingular scalar mixing matrix. The mixing is assumed to be instantaneous, so that there is no time delay between the source signals and the sensor observations.

To solve Equation (14.2) using the ICA algorithms, it is assumed that the source signals $\mathbf{s}(t)$ are mutually independent at each time instant t. This assumption is not unrealistic in many cases, and it need not be exactly true in practice, since the estimation results can provide a good approximation of the real source signals [15]. In this sense, the problem is to determine a constant (weight) matrix $\mathbf{W}$ so that $\hat{\mathbf{S}}$, an estimate of the source matrix, is as independent as possible:

$$\hat{\mathbf{S}} = \mathbf{WX} \tag{14.3}$$

In theory, the unmixing matrix $\mathbf{W}_{m\times n}$ can be solved using the Moore–Penrose pseudo-inverse of the mixing matrix $\mathbf{A}$

$$\mathbf{W} = (\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1}\mathbf{A}^{\mathrm{T}} \tag{14.4}$$

Correspondingly, the estimation of one independent component (one row of $\hat{\mathbf{S}}$) can be denoted as $\hat{\mathbf{s}}_i = \mathbf{wX}$, where $\mathbf{w}$ is one row of the unmixing matrix $\mathbf{W}$. Define $\mathbf{z} = \mathbf{A}^{\mathrm{T}}\mathbf{w}^{\mathrm{T}}$, then the independent component $\hat{\mathbf{s}}_i = \mathbf{wX} = \mathbf{wAS} = \mathbf{z}^{\mathrm{T}}\mathbf{S}$, which is a linear combination of the $\mathbf{s}_i$ values with the weights given by $\mathbf{z}$. According to the central limit theorem, the distribution of a sum of independent random variables converges to a Gaussian. Thus, $\mathbf{z}^{\mathrm{T}}\mathbf{S}$ is more Gaussian than any of the components $\mathbf{s}_i$ and becomes least Gaussian when it in fact equals one of the $\mathbf{s}_i$ values, i.e. when it gives the correct estimation of one of the sources [15]. Therefore, in the context of ICA, it is claimed that non-Gaussianity indicates independence. Many metrics have been studied to measure the non-Gaussianity of the independent components, such as kurtosis [13,22], mutual information [11,23], and negentropy [14,24].

For the linear mixing and unmixing models, it is assumed that at most one source signal is normally distributed [17]. This is because the mixture of two or more Gaussian sources is still a Gaussian, which makes the unmixing problem ill-posed. This assumption is reasonable in practice, since pure Gaussian processes are rare in real data.

© 2005 by Chapman & Hall/CRC

14.3 Source Number Estimation

The "blind" qualification of BSS assumes that there is no *a priori* information available on the number of sources, the distribution of sources, and the mixing model. Therefore, the BSS problem involves two implications: source number estimation and source separation. For conceptual and computational simplicity, most ICA algorithms assume the number of sources is equal to the number of observations, so that the mixing matrix $\mathbf{A}$ and the unmixing matrix $\mathbf{W}$ are square and form an inverse pair up to a scaling and permutation operation, which are easy to estimate. However, this equality assumption is not appropriate in sensor network applications since, in general, there are far more sensor nodes deployed than the targets. Hence, the number of targets has to be estimated before any further operations can be done.

Suppose $\mathcal{H}_m$ denotes the hypothesis on the number of sources m; the goal of source number estimation is to find $\hat{m}$ whose corresponding hypothesis $\mathcal{H}_{\hat{m}}$ maximizes the posterior probability given only the observation matrix $\mathbf{X}$:

$$\hat{m} = \arg\max_m P(\mathcal{H}_m|\mathbf{X}) \tag{14.5}$$

In the case that the number of observations is greater than the number of sources ($n > m$), several approaches have been developed, some heuristic, others based on more principled approaches. As discussed in Ref. [18], in recent years it has become clear that techniques of the latter category are superior and, at best, heuristic methods may be seen as approximations to some more detailed underlying principles. A brief introduction is given here on some principled source number estimation methods.

14.3.1 Bayesian Source Number Estimation

Roberts [21] proposed a Bayesian source number estimation approach in 1998 to find the hypothesis that maximizes the posterior probability $P(\mathcal{H}_m|\mathbf{X})$. Interested readers are referred these for a detailed theoretical derivation.

According to Bayes' theorem, the posterior probability of the hypothesis can be written as

$$P(\mathcal{H}_m|\mathbf{X}) = \frac{p(\mathbf{X}|\mathcal{H}_m)P(\mathcal{H}_m)}{p(\mathbf{X})} \tag{14.6}$$

Assume that the hypothesis $\mathcal{H}_m$ of different number of sources m has a uniform distribution, i.e. equal prior probability $P(\mathcal{H}_m)$. Since $p(\mathbf{X})$ is a constant, the measurement of the posterior probability can be simplified to the calculation of the likelihood $p(\mathbf{X}|\mathcal{H}_m)$. By marginalizing the likelihood over the system parameters space and approximating the marginal integrals by the Laplace approximation method, a log-likelihood function proportional to the posterior probability can be written as

$$\begin{aligned} L(m) &= \log p(\mathbf{x}(t)|\mathcal{H}_m) \\ &= \log \pi(\hat{\mathbf{s}}(t)) + \frac{1}{2}(n-m)\ \log\left(\frac{\hat{\beta}}{2\pi}\right) - \frac{1}{2}\ \log|\hat{\mathbf{A}}^{\mathrm{T}}\hat{\mathbf{A}}| - \frac{\hat{\beta}}{2}(\mathbf{x}(t) - \hat{\mathbf{A}}\hat{\mathbf{s}}(t))^2 \\ &\quad - \left\{\frac{mn}{2}\ \log\left(\frac{\hat{\beta}}{2\pi}\right) + \frac{n}{2}\left[\sum_{j=1}^{m}\log\hat{\mathbf{s}}_j(t)^2\right] + mn\log\gamma\right\} \end{aligned} \tag{14.7}$$

© 2005 by Chapman & Hall/CRC

where $\mathbf{x}(t)$ is the sensor observations, $\hat{\mathbf{A}}$ is the estimate of the mixing matrix, $\hat{\mathbf{s}}(t) = \mathbf{W}\mathbf{x}(t)$ is the estimate of the independent sources and $\mathbf{W} = (\hat{\mathbf{A}}^{\mathrm{T}}\hat{\mathbf{A}})^{-1}\hat{\mathbf{A}}^{\mathrm{T}}$, $\hat{\beta}$ is the variance of noise component, γ is a constant, and $\pi(\cdot)$ is the assumed marginal distribution of the sources.

The Bayesian source number estimation method considers a set of Laplace approximations to infer the posterior probabilities of specific hypotheses. This approach has a solid theoretical background and the objective function is easy to calculate; hence, it provides a practical solution for the source number estimation problem.

14.3.2 Sample-Based Source Number Estimation

Other than the Laplace approximation method, the posterior probabilities of specific hypotheses can also be evaluated using a sample-based approach. In this approach, a reversible-jump Markov chain Monte Carlo (RJ-MCMC) method is proposed to estimate the joint density over the mixing matrix $\mathbf{A}$, the hypothesized number of sources m, and the noise component R_{n}, which is denoted as $P(\mathbf{A}, m, R_{\mathrm{n}})$ [18,20]. The basic idea is to construct a Markov chain which generates samples from the hypothesis probability and to use the Monte Carlo method to estimate the posterior probability from the samples. An introduction of Monte Carlo methods can be found in Ref. [25].

RJ-MCMC is actually a random-sweep Metropolis–Hastings method, where the transition probability of the Markov chain from state $(\mathbf{A}, m, R_{\mathrm{n}})$ to state $(\mathbf{A}', m', R'_{\mathrm{n}})$ is

$$p = \min\left\{1,\ \frac{P(\mathbf{A}', m', R'_{\mathrm{n}}|\mathbf{X})}{P(\mathbf{A}, m, R_{\mathrm{n}}|\mathbf{X})}\frac{q(\mathbf{A}, m, R_{\mathrm{n}}|\mathbf{X})}{q(\mathbf{A}', m', R'_{\mathrm{n}}|\mathbf{X})}J\right\} \tag{14.8}$$

where $P(\cdot)$ is the posterior probability of the unknown parameters of interest, $q(\cdot)$ is a proposal density with each element drawn from a normal distribution with zero mean, and J is the ratio of Jacobians for the proposal transition between the two states [18]. More detailed derivation of this method is provided in [20].

14.3.3 Variational Learning

In recent years, the Bayesian inference problem shown in Equation (14.6) is also tackled using another approximative method known as variational learning [26,27]. In ICA problems, variables are divided into two classes: the visible variables v, such as the observation matrix $\mathbf{X}$; and the hidden variables h, such as an ensemble of the parameters of $\mathbf{A}$, the noise covariance matrix, any parameters in the source density models, and all associated hyperparameters, including the number of sources m [18]. Suppose $q(h)$ denotes the variational approximation to the posterior probability of the hidden variables $P(h|v)$, the negative variational free energy F is defined as

$$F = \int q(h)\ \ln\ P(h|v)\,\mathrm{d}h + H[q(h)] \tag{14.9}$$

where $H[q(h)]$ is the differential entropy of $q(h)$. It is shown that the negative free energy F forms a strict lower bound on the evidence of the model, with the difference being the Kullback–Leibler (KL) divergence between the true and approximating posteriors [28]. Therefore, maximizing F is equivalent to minimizing the KL divergence, and this process provides a direct means of source number estimation.

Another promising source number estimation approach using variational learning is the so-called automatic relevance determination (ARD) scheme [28]. The basic idea of ARD is to suppress sources that are unsupported by the data. For example, assume each hypothesized source has a Gaussian prior with separate variances, those sources that do not contribute to modeling the observations tend to have

© 2005 by Chapman & Hall/CRC

very small variances, and the corresponding source models do not move significantly from their priors [18]. After eliminating those unsupported sources, the sustained sources give the true number of sources of interest.

Even though variational learning is a particularly powerful approximative approach, it is yet to be developed into a more mature form. In addition, it presents difficulties in estimating the true number of sources with noisy data.

14.4 Distributed Source Number Estimation

The source number estimation algorithms described in Section 14.3 are all centralized processes, in the sense that the observation signals from all the sensor nodes are collected at a processing center and the estimation needs to be performed on the whole data set. While this assumption works well for small sensor-array applications, like in speech analysis, it is not necessarily the case for real-time applications in sensor networks due to the sheer amount of sensor nodes, the extremely constrained resource, and scalability issues.

The sensor nodes in sensor networks are usually battery supported and cannot be recharged in real time. Therefore, energy is the most constraining resource in sensor networks. It has been shown [29] that, among all the activities conducted on the sensor node, the wireless communication consumes the most energy. Hence, the centralized scheme in which all data are transmitted from each sensor node to a central processor to form a large data set for source number estimation will consume too much energy and is not an option for real-time sensor network applications. On the contrary, when implemented in a distributed manner, data can be processed locally on a cluster of sensor nodes that are close in geographical location and only local decisions need to be transfered for further processing. In this way, the distributed target detection framework can dramatically reduce long-distance network traffic and, therefore, conserve the energy consumed on data transmissions and prolong the lifetime of the sensor network.

14.4.1 Distributed Hierarchy in Sensor Networks

In the context of the proposed distributed solution to the source number estimation problem, we assume a clustering protocol has been applied and the sensor nodes have organized themselves into clusters with each node assigned to one and only one cluster. Local nodes can communicate with other nodes within the same cluster, and different clusters, communicate through a cluster head specified within each cluster. An example of a clustered sensor network is illustrated in Figure 14.1.

Suppose there are m targets present in the sensor field, and the sensor nodes are divided into L clusters. Each cluster l $(l = 1, \ldots, L)$ can sense the environment independently and generate an observation matrix $\mathbf{X}_l$ which consists of mixtures of the source signals generated by the m targets. The distributed estimation hierarchy includes two levels of processing. First, the posterior probability of each hypothesis $\mathcal{H}_m$ on the number of sources m given an observation matrix $\mathbf{X}_l$, $P(\mathcal{H}_m|\mathbf{X}_l)$, is estimated within each cluster l. The Bayesian source number estimation approach proposed by Roberts [21] is employed in this step. Second, the decisions from each cluster are fused using an *a posteriori* probability fusion algorithm. The structure of the hierarchy is illustrated in Figure 14.2.

The developed distributed source number estimation hierarchy benefits from two research avenues: distributed detection and ICA model order estimation. However, it exhibits some unique features that make it suitable for multiple target detection in sensor networks from both the theoretical and practical points of view.

- M-ary hypothesis testing. Most distributed detection algorithms are derived under the binary hypothesis assumption, where $\mathcal{H}$ takes on one of two possible values corresponding to the presence or absence of the target [30]. The distributed framework developed here extends the traditional binary hypothesis testing problem to the M-ary case, where the values of $\mathcal{H}$ correspond to the different numbers of sources.

© 2005 by Chapman & Hall/CRC

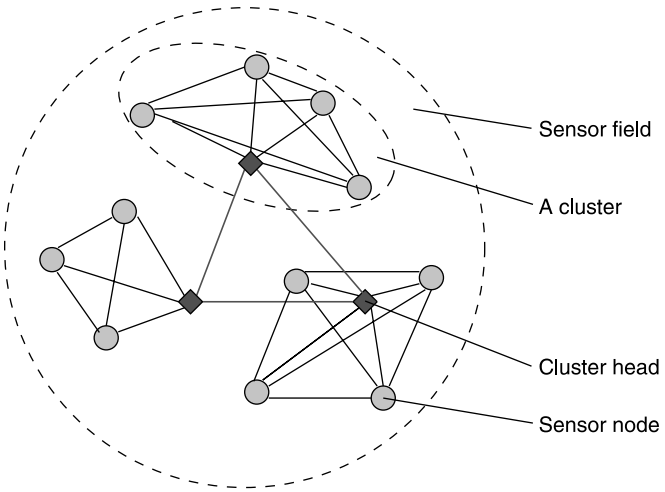


Figure 14.1. An example of a clustered sensor network.

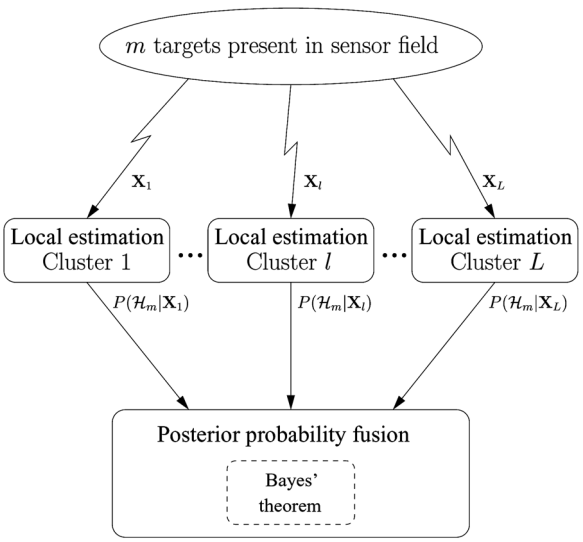


Figure 14.2. The structure of the distributed source number estimation hierarchy.

- Fusion of detection probabilities. Instead of making a crisp decision from local cluster estimations as in the classic distributed detection algorithms, a Bayesian source number estimation algorithm is performed on the observations from each cluster, and the *a posteriori* probability for each hypothesis is estimated. These probabilities are then sent to a fusion center where a decision regarding the source number hypothesis is made. This process is also referred to as the fusion of detection probabilities [31] or combination of level of significance [32]. By estimating and fusing the probabilities of hypothesis from each cluster, it is possible for the system to achieve a higher detection accuracy.
- Distributed structure. Even though the source number estimation can also be implemented in a centralized manner, where the signals captured by all the sensor nodes are transferred to a processing center and the estimation is performed on the whole data set, the distributed

© 2005 by Chapman & Hall/CRC

framework presents several advantages that make it more appropriate for real-time sensor network applications. For example, in the distributed framework, data are processed locally in each cluster and only the estimated hypothesis probabilities are transmitted through the network. Hence, it can reduce the long-distance network traffic significantly, and consequently, conserve energy. Furthermore, since the estimation process is performed in parallel within each cluster, the computation burden is distributed and computation time reduced.

After local source number estimation is conducted within each cluster, a posterior probability fusion method based on Bayes' theorem is derived to fuse the results from each cluster.

14.4.2 Posterior Probability Fusion Based on Bayes' Theorem

The objective of the source number estimation approaches is to find the optimal number of sources m that maximizes the posterior probability $P(\mathcal{H}_m|\mathbf{X})$. When implemented in the distributed hierarchy, the local estimation approach calculates the posterior probability corresponding to each hypothesis $\mathcal{H}_m$ from each cluster $P(\mathcal{H}_m|\mathbf{X}_1), \ldots, P(\mathcal{H}_m|\mathbf{X}_L)$, where L is the number of clusters. According to Bayes' theorem, the fused posterior probability can be written as

$$P(\mathcal{H}_m|\mathbf{X}) = \frac{p(\mathbf{X}|\mathcal{H}_m)P(\mathcal{H}_m)}{p(\mathbf{X})} \tag{14.10}$$

Assume the clustering of sensor nodes is exclusive, i.e. $\mathbf{X} = \mathbf{X}_1 \cup \mathbf{X}_2 \cup \cdots \cup \mathbf{X}_L$ and $\mathbf{X}_l \cap \mathbf{X}_q = \emptyset$ for any $l \neq q$, $l = 1, \ldots, L$, $q = 1, \ldots, L$, the posterior probability $P(\mathcal{H}_m|\mathbf{X})$ can be represented as

$$P(\mathcal{H}_m|\mathbf{X}) = \frac{p(\mathbf{X}_1 \cup \mathbf{X}_2 \cup \cdots \cup \mathbf{X}_L|\mathcal{H}_m)P(\mathcal{H}_m)}{p(\mathbf{X}_1 \cup \mathbf{X}_2 \cup \cdots \cup \mathbf{X}_L)} \tag{14.11}$$

Since the observations from different clusters $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_L$ are assumed to be independent, $p(\mathbf{X}_l \cap \mathbf{X}_q) = 0$, for any $l \neq q$, we then have

$$\begin{aligned} p(\mathbf{X}_1 \cup \mathbf{X}_2 \cup \cdots \cup \mathbf{X}_L|\mathcal{H}_m) &= \sum_{l=1}^{L} p(\mathbf{X}_l|\mathcal{H}_m) - \sum_{l,q=1,l\neq q}^{L} p(\mathbf{X}_l \cap \mathbf{X}_q|\mathcal{H}_m) \\ &= \sum_{l=1}^{L} p(\mathbf{X}_l|\mathcal{H}_m) \end{aligned} \tag{14.12}$$

Combining Equations (14.11) and (14.12), the fused posterior probability can be calculated as

$$\begin{aligned} P(\mathcal{H}_m|\mathbf{X}) &= \frac{\sum_{l=1}^{L} p(\mathbf{X}_l|\mathcal{H}_m)P(\mathcal{H}_m)}{\sum_{l=1}^{L} p(\mathbf{X}_l)} \\ &= \frac{\sum_{l=1}^{L} \frac{P(\mathcal{H}_m|\mathbf{X}_l)p(\mathbf{X}_l)}{P(\mathcal{H}_m)} P(\mathcal{H}_m)}{\sum_{l=1}^{L} p(\mathbf{X}_l)} \\ &= \sum_{l=1}^{L} \frac{p(\mathbf{X}_l)}{\sum_{q=1}^{L} p(\mathbf{X}_q)} P(\mathcal{H}_m|\mathbf{X}_l) \end{aligned} \tag{14.13}$$

© 2005 by Chapman & Hall/CRC

where $P(\mathcal{H}_m|\mathbf{X}_l)$ denotes the posterior probability calculated in cluster l, and the term $p(\mathbf{X}_l)/\sum_{q=1}^{L} p(\mathbf{X}_q)$ reflects the physical characteristic of the clustering in the sensor network, which is application specific. For example, in the case of distributed multiple target detection using acoustic signals, the propagation of acoustic signals follows the energy decay model, that the energy detected is inversely proportional to the square of the distance between the source and the sensor node, i.e., $E_{\text{sensor}} \propto 1/d^2 E_{\text{source}}$. Therefore, the term $p(\mathbf{X}_l)/\sum_{q=1}^{L} p(\mathbf{X}_q)$ can be considered as the relative detection sensitivity of the sensor nodes in cluster l and is proportional to the average energy captured by the sensor nodes:

$$\frac{p(\mathbf{X}_l)}{\sum_{q=1}^{L} p(\mathbf{X}_q)} \propto \frac{1}{K_l}\sum_{k=1}^{K_l} E_k \propto \frac{1}{K_l}\sum_{k=1}^{K_l}\frac{1}{d_k^2} \tag{14.14}$$

where K_l denotes the number of sensor nodes in cluster l.

14.5 Performance Evaluation

We apply the proposed distributed source number estimation hierarchy to detect multiple civilian targets using data collected from a field demo held at BAE Systems, Austin, TX, in August 2002. We also compare the performance between the centralized Bayesian source number estimation algorithm and the distributed hierarchy using the evaluation metrics described below.

14.5.1 Evaluation Metrics

As mentioned before, the source number estimation is basically an optimization problem in which an optimal hypothesis $\mathcal{H}_m$ is pursued that maximizes the posterior probability given the observation matrix, $P(\mathcal{H}_m|\mathbf{X})$. The optimization process is affected by the initialization condition and the update procedure of the algorithm itself. To compensate for the randomness and to stabilize the overall performance, the algorithms are performed repeatedly, e.g. 20 times in this experiment.

The detection probability $P_{\text{detection}}$ is the most intuitive metric to measure the accuracy of a detection approach. It is defined as the ratio between the correct source number estimations and the total number of estimations, i.e. $P_{\text{detection}} = N_{\text{correct}}/N_{\text{total}}$, where N_{correct} denotes the number of correct estimations and N_{total} is the total number of estimations.

After repeating the algorithm for multiple times, we can then generate a histogram that shows the accumulated number of estimations corresponding to different hypotheses of the number of sources. The histogram also represents the reliability of the algorithm, in the sense that the greater the difference of the histogram values between the hypothesis of the correct estimate and other hypotheses, then the more deterministic and reliable the algorithm. We use kurtosis β to measure this characteristic of the histogram. Kurtosis calculates the flatness of the histogram,

$$\beta = \frac{1}{C}\sum_{k=1}^{N} k\left(\frac{h_k - \mu}{\theta}\right)^4 - 3 \tag{14.15}$$

where h_k denotes the value of the kth bin in the histogram, N is the total number of bins, $C = \sum_{k=1}^{N} h_k$, $\mu = (1/C)\sum_{k=1}^{N} k h_k$ is the mean, and $\theta = \sqrt{(1/C)\sum_{k=1}^{N} k(h_k - \mu)^2}$ is the variance. Intuitively, the larger the kurtosis, the more deterministic the algorithm, and the more reliable the estimation.

Since the source number estimation is designed for real-time multiple target detection in sensor networks, the computation time is also an important metric for performance evaluation.

© 2005 by Chapman & Hall/CRC

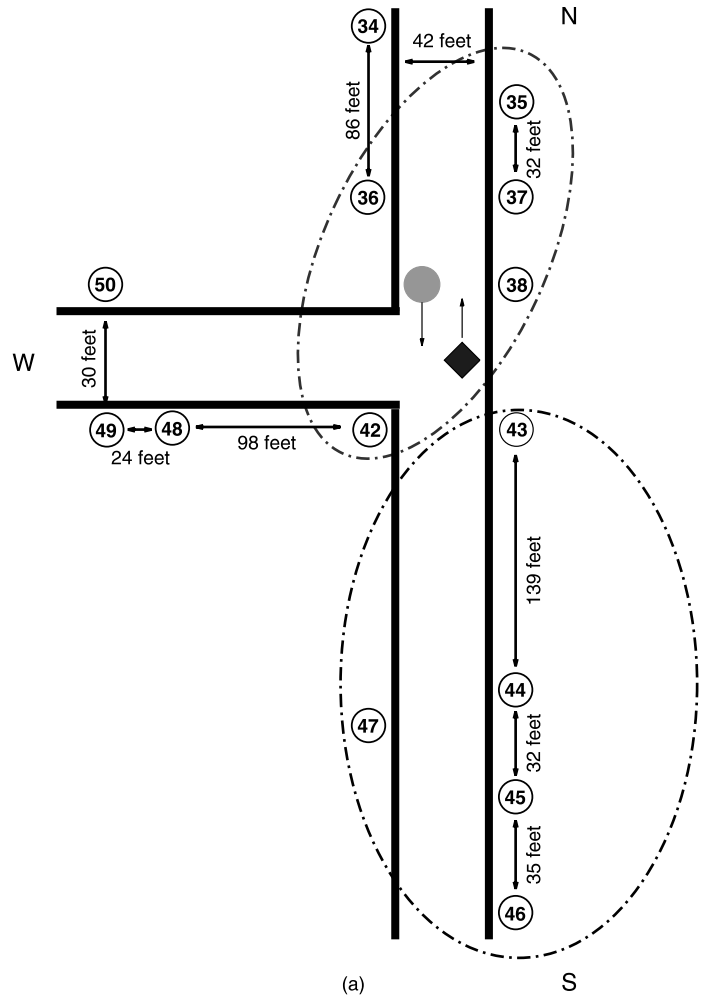
14.5.2 Experimental Results

In the field demo, we let two civilian vehicles, a motorcycle and a diesel truck, as shown in Figure 14.3, travel along the N–S road from opposite directions and intersect at the T-junction. There are 15 nodes deployed along the road. For this experiment, we assume that two clusters of five sensor nodes exist for the distributed processing. The sensor network setup is illustrated in Figure 14.4(a). We use the Sensoria WINS NG-2.0 sensor nodes [as shown in Figure 14.4(b)], which consist of a dual-issue SH-4 processor running at 167 MHz with 300 MIPS of processing power, radio-frequency modem for wireless communication, and up to four channels of sensing modalities, such as acoustic, seismic, and infrared. In this experiment, we perform the multiple target detection algorithms on the acoustic signals captured by the microphone on each sensor node. The observations from sensor nodes are preprocessed component-wise to be zero-mean, unit-variance distributed.



Figure 14.3. Vehicles used in the experiment: (a) Motorcycle; (b) diesel truck.

© 2005 by Chapman & Hall/CRC



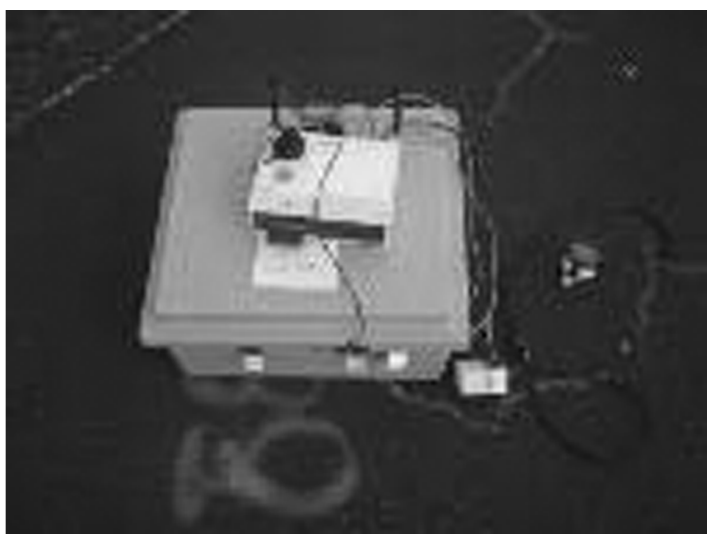


Figure 14.4. The sensor laydown (a) and the Sensoria sensor node (b) used [33].

© 2005 by Chapman & Hall/CRC

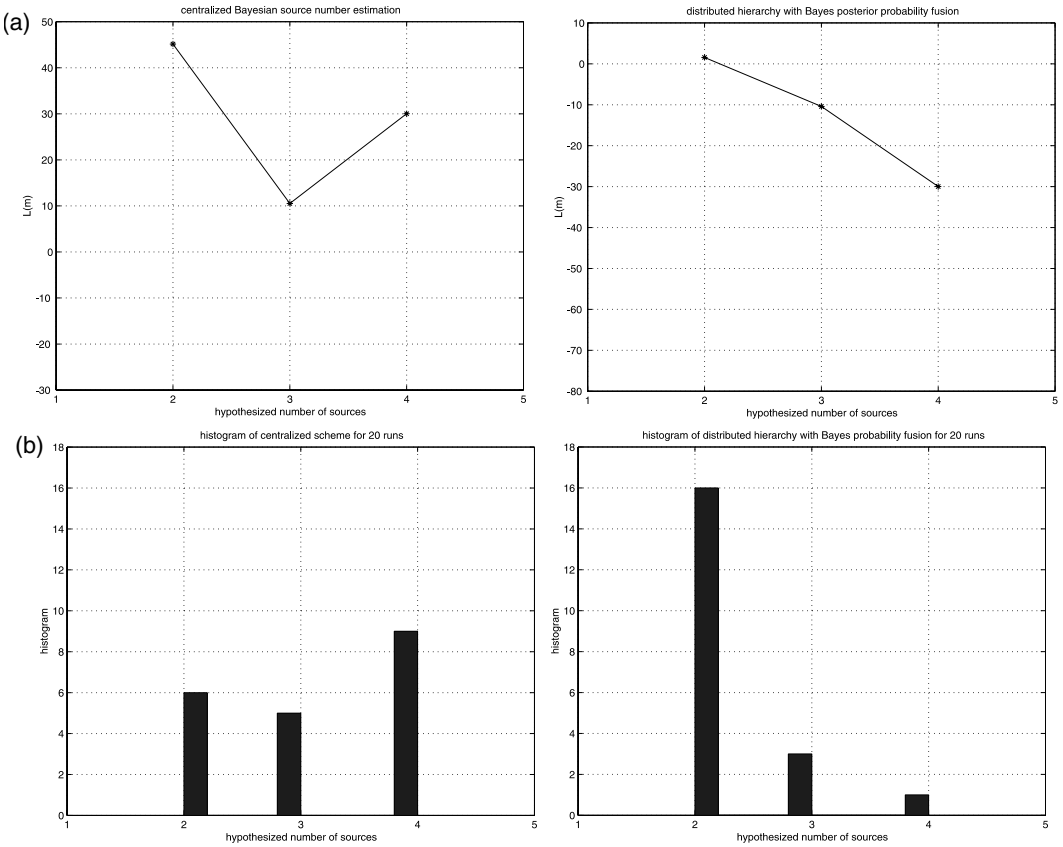


Figure 14.5. Performance comparison: (a) log-likelihood function; (b) histogram of source number estimation during 20 repetitions. Left: centralized Bayesian approach. Right: distributed hierarchy with the Bayesian posterior probability fusion. (Figure taken from Wang, X. *et al.*, Distributed source number estimation for multiple target detection in sensor networks, *IEEE Workshop on Statistical Signal Processing*, St. Louis, MO, September 28–October 1, 2003, 395, © 2003 IEEE.)

First, the centralized Bayesian source number estimation algorithm is performed using all ten of the sensor observations. Second, the distributed hierarchy is applied as shown in Figure 14.2, which first calculates the corresponding posterior probabilities of different hypotheses in the two clusters and then fuses the local results using the Bayesian posterior probability fusion method.

Figure 14.5(a) shows the average value of the log-likelihood function in Equation (14.7) corresponding to different hypothesized numbers of sources over 20 repetitions. Figure 14.5(b) displays the histogram of the occurrence of the most probable number of sources when the log-likelihood function is evaluated twenty times. Each evaluation randomly initializes the mixing matrix $\mathbf{A}$ with values drawn from a zero-mean, unit-variance normal distribution. The left column in the figure corresponds to the performance of applying the centralized Bayesian source number estimation approach on all ten of the sensor observations. The right column shows the corresponding performance of the distributed hierarchy with the proposed Bayesian posterior probability fusion method. Based on the average log-likelihood, it is clear that in both approaches the hypothesis with the true number of sources ($m = 2$) has the greatest support. However, with the algorithms being performed for 20 times, they present different rates of correct estimations and different levels of uncertainty.

Figure 14.6(a) illustrates the kurtosis calculated from the two histograms in Figure 14.5(b). The larger the kurtosis, the more deterministic the result, and the more reliable the approach. We can see that the

© 2005 by Chapman & Hall/CRC

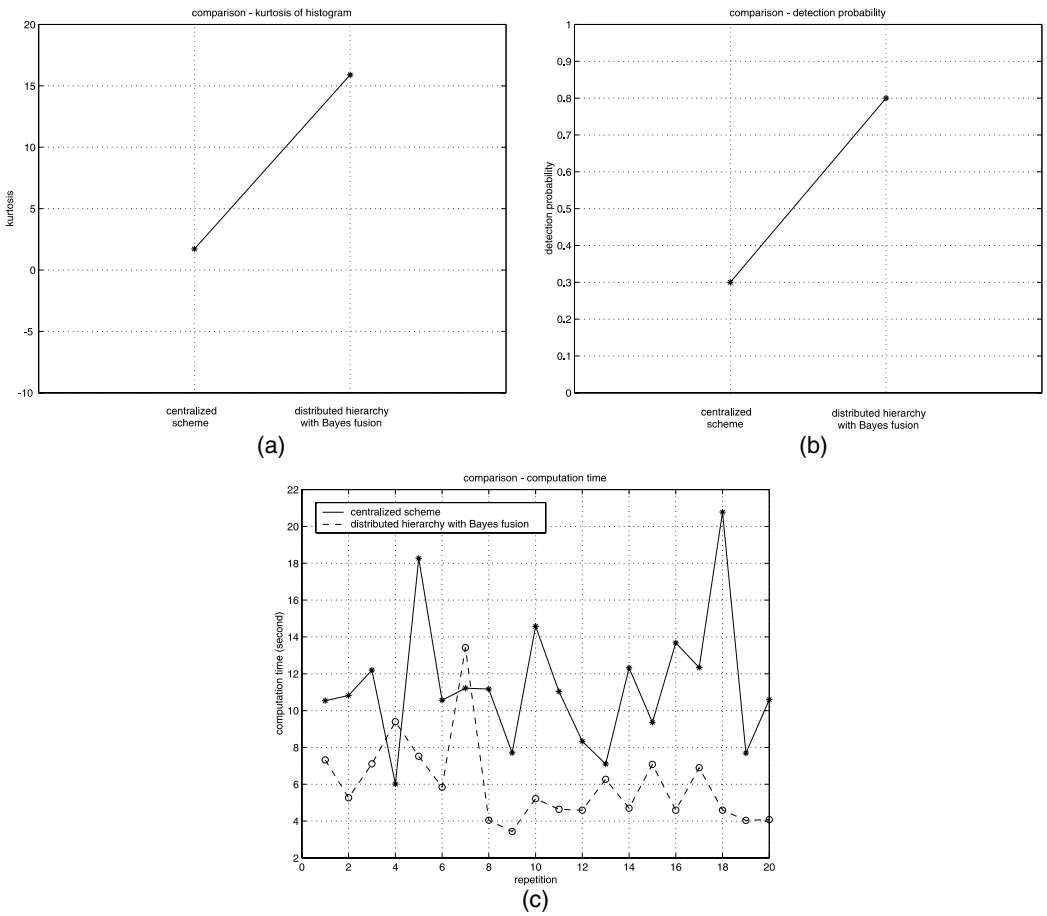


Figure 14.6 Comparison: (a) kurtosis, (b) detection probability, and (c) computation time.

kurtosis of the distributed approach is eight times higher than that of the centralized approach. The detection probabilities are shown in Figure 14.6(b). We observe that the centralized Bayesian algorithm can detect the correct number of sources 30% of the time, whereas the distributed approach increases the number of correct estimates by an average of 50%. The comparison of the computation times during the 20 runs between the centralized scheme and the distributed hierarchy is shown in Figure 14.6(c). It is clear that by using the distributed hierarchy, the computation time is generally reduced by a factor of 2.

14.5.3 Discussion

As demonstrated in the experiment, as well as in the performance evaluation, the distributed hierarchy with the proposed Bayesian posterior probability fusion method has better performance, in the sense that it can provide higher detection probability, can be more deterministic, and is reliable. The reasons include: (1) The centralized scheme uses the observations from all the sensor nodes as inputs to the Bayesian source number estimation algorithm. The algorithm is thus sensitive to signal variations due to node failure or environmental noise in each input signal. In the distributed framework, however, the source number estimation algorithm is only performed within each cluster; therefore, the effect of signal variations is limited locally and might contribute less in the posterior probability fusion process.

© 2005 by Chapman & Hall/CRC

(2) In the derivation of the Bayesian posterior probability fusion method, the physical characteristics of sensor networks, such as the signal energy captured by each sensor node versus its geographical position, are considered, making this method more adaptive to real applications.

Furthermore, the distributed hierarchy is able to reduce the network traffic by avoiding long-distance data transmission, hence conserving energy and providing a scalable solution. The parallel implementation of the estimation algorithm in each cluster can also reduce the computation time by half.

14.6 Conclusions

This work studied the problem of source number estimation in sensor networks for multiple target detection. This problem is similar to the BSS problem in signal processing, and ICA is the most popular algorithm to solve it. The classical BSS problem includes two implications: source number estimation and source separation. We consider that the multiple target detection in sensor networks follows the same principle as the source number estimation problem. We first summarized several centralized source number estimation approaches. However, sensor networks usually consist of thousands of sensor nodes that are deployed densely in the field, and each sensor node only has a limited power supply. Hence, the source estimation algorithm has to be operated in a distributed manner in order to avoid large amount of long-distance data transmission. This, in turn, reduces the network traffic and conserves energy. A distributed source number estimation hierarchy in sensor networks is developed in this chapter. This includes two levels of processing. First, a local source number estimation is performed in each cluster using the centralized Bayesian source number estimation approach. Then a posterior probability fusion method is derived based on Bayes' theorem to combine the local estimations and generate a global decision. An experiment is conducted on the detection of multiple civilian vehicles using acoustic signals to evaluate the performance of the approaches. The distributed hierarchy with the Bayesian posterior probability fusion method is shown to provide better performance in terms of the detection probability and reliability. In addition, the distributed framework can reduce the computation time by half.

Acknowledgments

Figures 14.4(a) and 14.5 are taken from [33]. (c) 2003 IEEE. Reprinted with permission.

References

[1] Akyildiz, I.F. *et al.*, A survey on sensor networks, *IEEE Communications Magazine*, 40(8), 102, 2002.
[2] Kumar, S. *et al.*, Collaborative signal and information processing in micro-sensor networks, *IEEE Signal Processing Magazine*, 19(2), 13, 2002.
[3] Li, D. *et al.*, Detection, classification, and tracking of targets, *IEEE Signal Processing Magazine*, 19(2), 17, 2002.
[4] Wang, X. *et al.*, Collaborative multi-modality target classification in distributed sensor networks, in *Proceedings of the Fifth International Conference on Information Fusion*, Annapolis, MD, July 2002, vol. 1, 285.
[5] Yao, K. *et al.*, Maximum-likelihood acoustic source localization: experimental results, in *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2002, vol. 3, 2949.
[6] Zhao, F. *et al.*, Information-driven dynamic sensor collaboration, *IEEE Signal Processing Magazine*, 19(2), 61, 2002.
[7] Knaian, A.N., A wireless sensor network for smart roadbeds and intelligent transportation systems, M.S. thesis, Massachusetts Institute of Technology, June 2000.
[8] Delin, K.A. and Jackson, S.P., Sensor web for *in situ* exploration of gaseous biosignatures, in *Proceedings of 2000 IEEE Aerospace Conference*, Big Sky, MT, March 2000.

© 2005 by Chapman & Hall/CRC

[9] Yang, X. *et al.*, Design of a wireless sensor network for long-term, *in-situ* monitoring of an aqueous environment, *Sensors*, 2(7), 455, 2002.
[10] Cerpa, A. *et al.*, Habitat monitoring: application driver for wireless communications technology, in *2001 ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean*, April, 2001.
[11] Bell, A.J. and Sejnowski, T.J., An information-maximisation approach to blind separation and blind deconvolution, *Neural Computation*, 7(6), 1129, 1995.
[12] Herault, J. and Jutten, J., Space or time adaptive signal processing by neural network models, in *Neural Networks for Computing: AIP Conference Proceedings 151*, Denker, J.S. (ed.), American Institute for Physics, New York, 1986.
[13] Tan, Y. and Wang, J., Nonlinear blind source separation using higher order statistics and a genetic algorithm, *IEEE Transactions on Evolutionary Computation*, 5(6), 600, 2001.
[14] Comon, P., Independent component analysis, a new concept, *Signal Processing*, 36(3), 287, April 1994.
[15] Hyvarinen, A. and Oja, E., Independent component analysis: a tutorial, http://www.cis.hut.fi/aapo/papers/IJCNN99_tutorialweb/, April 1999.
[16] Karhunen, J., Neural approaches to independent component analysis and source separation, in *Proceedings of 4th European Symposium on Artificial Neural Networks (ESANN)*, 249, 1996.
[17] Lee, T. *et al.*, A unifying information-theoretic framework for independent component analysis, *International Journal on Mathematical and Computer Modeling*, 39, 1, 2000.
[18] Roberts, S. and Everson, R. (eds.), *Independent Component Analysis: Principles and Practice*, Cambridge University Press, 2001.
[19] Knuth, K.H., A Bayesian approach to source separation, in *Proceedings of First International Conference on Independent Component Analysis and Blind Source Separation: ICA'99*, 283, 1999.
[20] Richardson, S. and Green, P.J., On B ayesian analysis of mixtures with an unknown number of components, *Journal of the Royal Statistical Society, Series B*, 59(4), 731, 1997.
[21] Roberts, S.J., Independent component analysis: source assessment & separation, a Bayesian approach, *IEE Proceedings on Vision, Image, and Signal Processing*, 145(3), 149, 1998.
[22] Hyvarinen, A. and Oja, E., A fast fixed-point algorithm for independent component analysis, *Neural Computation*, 9, 1483, 1997.
[23] Linsker, R., Local synaptic learining rules suffice to maximize mutual information in a linear network, *Neural Computation*, 4, 691, 1992.
[24] Hyvarinen, A., Fast and robust fixed-point algorithms for independent component analysis, *IEEE Transactions on Neural Networks*, 10(3), 626, 1999.
[25] MacKay, D.J.C., Monte Carlo methods, in *Learning in Graphical Models*, Jordan, M.I. (ed.), Kluwer, 175, 1999.
[26] Attias, H., Inferring parameters and structure of latent variable models by variational Bayes, in *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, 21, 1999.
[27] Bishop, C.M., *Neural Networks for Pattern Recognition*, Oxford University Press, 1995.
[28] Choudrey, R. *et al.*, An ensemble learning approach to independent component analysis, in *Proceedings of Neural Networks for Signal Processing*, Sydney, 2000.
[29] Raghunathan, V. *et al.*, Energy-aware wireless microsensor networks, *IEEE Signal Processing Magazine*, 19(2) 40, March 2002.
[30] Chamberland, J. and Veeravalli, V.V., Decentralized detection in sensor networks, *IEEE Transactions on Signal Processing*, 51(2), 407, February 2003.
[31] Krysztofowicz, R. and Long, D., Fusion of detection probabilities and comparison of multisensor systems, *IEEE Transactions on System, Man, and Cybernetics*, 20, 665, May/June 1990.
[32] Hedges, V. and Olkin, I., *Statistical Methods for Meta-Analysis*, Academic Press, New York, 1985.
[33] Wang, X., Qi, H., Du, H. Distributed source number estimation for multiple target detection in sensor networks, *IEEE Workshop on Statistical Signal Processing*, St. Louis, MO, September 28–October 1, 2003, 395.

© 2005 by Chapman & Hall/CRC

III

Information Fusion

15. Foundations of Data Fusion for Automation
S.S. Iyengar, S. Sastry, and N. Balakrishnan *291*
Introduction • Automation Systems • Data Fusion Foundations • Security Management for Discrete Automation • Conclusions • Acknowledgements

16. Measurement-Based Statistical Fusion Methods For Distributed Sensor Networks *Nageswara S.V. Rao* *301*
Introduction • Classical Fusion Problems • Generic Sensor Fusion Problem • Empirical Risk Minimization • Statistical Estimators • Applications • Performance of Fused System • Metafusers • Conclusions • Acknowledgment

17. Soft Computing Techniques *R.R. Brooks* *321*
Problem Statement • Genetic Algorithms • Simulated Annealing • Trust • Tabu Search • Artificial Neural Networks • Fuzzy Logic • Linear Programming • Summary

18. Estimation and Kalman Filters *David L. Hall* *335*
Introduction • Overview of Estimation Techniques • Batch Estimation • Sequential Estimation and Kalman Filtering • Sequential Processing Implementation Issues

19. Data Registration *R.R. Brooks, Jacob Lamb, and Lynne Grewe* *361*
Problem Statement • Coordinate Transformations • Survey of Registration Techniques • Objective Functions • Results from Meta-Heuristic Approaches • Feature Selection • Real-Time Registration of Video Streams with Different Geometries • Summary

20. Signal Calibration, Estimation for Real-Time Monitoring and Control *Asok Ray and Shashi Phoha* *391*
Introduction • Signal Calibration and Measurement Estimation • Sensor Calibration in a Commercial-Scale Fossil-Fuel Power Plant

© 2005 by Chapman & Hall/CRC

• Summary and Conclusions • Appendix A: Multiple Hypotheses Testing Based on Observations of a Single Variable

21. Semantic Information Extraction *David S. Friedlander* *409*
Introduction • Symbolic Dynamics • Formal Language Measures • Behavior Recognition • Experimental Verification • Conclusions and Future Work • Acknowledgments and Disclaimer

22. Fusion in the Context of Information Theory *Mohiuddin Ahmed and Gregory Pottie* *419*
Introduction • Information Processing in Distributed Networks • Evolution Towards Information-Theoretic Methods for Data Fusion • Probabilistic Framework for Distributed Processing • Bayesian Framework for Distributed Multi-Sensor Systems • Concluding Remarks

23. Multispectral Sensing *N.K. Bose* *437*
Motivation • Introduction to Multispectral Sensing • Mathematical Model for Multisensor Array-Based Superresolution • Color Images • Conclusions • Acknowledgment

Once signals and images have been locally processed, additional work is performed to make global decisions from the local information. This section considers issues concerning information and data fusion. Fusion can occur at many different levels and in many different ways. The chapters in this section give an overview of the most important technologies.

Iyengar *et al.* provide a conceptual framework for data fusion systems. This approach is built upon two primary concepts:

(i) A mdoel describes the conceptual framework of the system. This is the structure of the global system.
(ii) A goal-seeking paradigm is used to guide the system in combining information.

Rao discusses the statistical concepts that underlie information fusion. Data is retrieved from noise-corrupted signals, and features inferred. The task is to extract information from sets of features that follow unknown statistical distributions. He provide a statistical discussion that unifies neural networks, vector-space and Nadaraya-Watson methods.

Brooks reviews soft computing methodologies that have been applied to information fusion. The methods discussed include the following families of meta-heuristics: genetic algorithms, simulated annealing, tabu search, artificial neural networks, TRUST, fuzzy logic, and linear programming. From this viewpoint, information fusion is phrased as an optimization problem. A solution is sought which minimizes a given objective function. Among other things, this function could be the amount of ambiguity in the system.

Hall provides an overview of estimation and Kalman filters. This approach ses control theoretical techniques. A system model is derived and an optimization approach is used to fit the data to the model. These approaches can provide optimal solutions to a large class of data fusion problems. The recursive nature of the Kalman filter algorithm has made it attractive for many real-time applications.

Brooks *et al.* tackle the problem of data registration. Readings from different sources must be mapped to a common coordinate system. This is a difficult problem, which is highly dependent on sensor geometry. This chapter provides extensive mathematical background and a survey of the best-known techniques. An example application is given using soft computing techniques. Other problems addressed include selecting proper features for registering images, and registering images from sensors with different geometries.

© 2005 by Chapman & Hall/CRC

Ray and Phoha provide a distributed sensor calibration approach. A set of sensors monitors an ongoing process. The sensor hardware will degrade over time, causing the readings to drift from the correct value. By comparing the readings over time and calculating the variance of the agreement, it is possible to assign trust values to the sensors. These values are then used to estimate the correct reading, allowing the sensors to be recalibrated online.

Friedlander uses an innovative approach to extract symbolic data from streams of sensor data. The symbols are then used to derive automata that describe the underlying process. In the chapter, he uses the derived automata to recognize complex behaviors of targets under surveillance. Of particular interest, is the fact that the abstraction process can be used to combine data of many different modes.

Ahmed and Pottie use information theory to analyze the information fusion problem. Distributed estimation and signal detection applications are considered. A Bayesian approach is provided and justified using information measures.

Bose considers the use of multispectral sensors. This class of sensors simultaneously collects image data using many different wavelengths. He describes the hardware used in multispectral sensing and how it is possible to achieve subpixel accuracy from the data.

This section provides a broad overview of information fusion technology. The problem is viewed from many different perspectives. Many different data modalities are considered, as are the most common applications of information fusion.

© 2005 by Chapman & Hall/CRC

15

Foundations of Data Fusion for Automation*

S.S. Iyengar, S. Sastry, and N. Balakrishnan

15.1 Introduction

Data fusion is a paradigm for integrating data from multiple sources to synthesize new information such that the whole is greater than the sum of its parts. This is a critical task in contemporary and future systems that are distributed networks of low-cost, resource-constrained sensors [1,2]. Current techniques for data fusion are based on general principles of distributed systems and rely on cohesive data representations to integrate multiple sources of data. Such methods do not extend easily to systems in which real-time data must be gathered periodically, by cooperative sensors, where some decisions become more critical than other decisions episodically.

There has been an extensive study in the areas of multi-sensor fusion and real-time sensor integration for time-critical sensor readings [3]. A distributed sensor data network is a set of spatially scattered sensors designed to derive appropriate inferences from the information gathered. The development of such networks for information gathering in unstructured environments is receiving a lot of interest, partly because of the availability of new sensor technology that is economically feasible to implement [4]. Sensor data networks represent a class of distributed systems that are used for sensing and *in situ* processing of spatially and temporally dense data from limited resources and harsh environments, by routing and cooperatively processing the information gathered. In all these systems, the critical step is the fusion of data gathered by sensors to synthesize new information.

Our interest is to develop data fusion paradigms for sensor–actuator networks that perform engineering tasks and we use Automation Systems as an illustrative example. automation systems represent an important, highly engineered, domain that has over a trillion dollars of installed base in the U.S. The real-time and distributed nature of these systems, with the attendant demands for safety, determinism, and predictability, represent significant challenges, and hence these systems are a good example. An Automation system is a collection of devices, equipment, and networks that regulate operations in a variety of manufacturing, material and people moving, monitoring, and safety applications.

*First published in *IEEE Instrumentation and Measurement Magazine*, 6(4), 35–41, 2003 and used with permission.

© 2005 by Chapman & Hall/CRC

Automation systems evolved from early centralized systems to large distributed systems that are difficult to design, operate, and maintain [5]. Current hierarchical architectures, the nature and use of human–computer-interaction (HCI) devices, and the current methods for addressing and configuration increase system life-cycle costs. Current methods to integrate system-wide data are hardcoded into control programs and not based on an integrating framework. Legacy architectures of existing automation systems are unable to support future trends in distributed automation systems [6]. Current methods for data fusion are also unlikely to extend to future systems because of system scale and simplicity of nodes.

We present a new integrating framework for data fusion that is based on two systems concepts: a conceptual framework and the goal-seeking paradigm [7]. The conceptual framework represents the structure of the system and the goal-seeking paradigm represents the behavior of the system. Such a systematic approach to data fusion is essential for proper functioning of future sensor–actuator networks [8] and SmartSpace [9]. In the short term, such techniques help to infuse emerging paradigms in to existing automation architectures. We must bring together knowledge in the fields of sensor fusion, data and query processing, automation systems design, and communication networks to develop the foundations. While extensive research is being conducted in these areas, as evidenced by the chapters compiled in this book, we hope that this special issue will open a window of opportunity for researchers in related areas to venture in to this emerging and important area of research [2].

15.2 Automation Systems

An automation system is a unique distributed real-time system that comprises a collection of sensors, actuators, controllers, communication networks, and user-interface devices. Such systems regulate the coordinated operation of physical machines and humans to perform periodic and precise tasks that may sometimes be dangerous for humans to perform. Examples of automation systems are: a factory manufacturing cars, a baggage handling system in an airport, and an amusement park ride. *Part*, *process*, and *plant* are three entities of interest. A plant comprises a collection of stations, mechanical fixtures, energy resources, and control equipment that regulate operations using a combination of mechanical, pneumatic, hydraulic, electric, and electronic components or subsystems. A process specifies a sequence of stations that a part must traverse through and operations that must be performed on the part at each station.

Figure 15.1 shows the major aspects of an automation system. All five aspects, namely input, output, logic processing, behavior specification, and HCI must be designed, implemented, and commissioned to operate an automation system successfully. Sensors and actuators are transducers that are used to acquire inputs and set outputs respectively. The controller periodically executes logic to determine new output values for actuators. HCI devices are used to specify logic and facilitate operator interaction at runtime.

The architecture of existing automation systems is hierarchical and the communication infrastructure is based on proprietary technologies that do not scale well. Ethernet is emerging as the principal control and data-exchange network. The transition from rigid, proprietary networks to flexible, open networks introduces new problems into the automation systems domain and security is a critical problem that demands attention.

15.2.1 System Characteristics

Automation systems operate in different modes. For example, *k*-hour-run is a mode that is used to exercise the system without affecting any parts. Other examples of modes are automatic, manual, and semi-automatic. In all modes, the overriding concern is to achieve deterministic, reliable, and safe operations. The mode of the system dictates the degree to which humans interact with the system. Safety checks performed in each of these modes usually decrease inversely with the degree of user interaction.

Communication traffic changes with operating mode. For example, when the system is in automatic mode, limited amounts of data (a few bytes) are exchanged; such data flows occur in localized areas of

© 2005 by Chapman & Hall/CRC

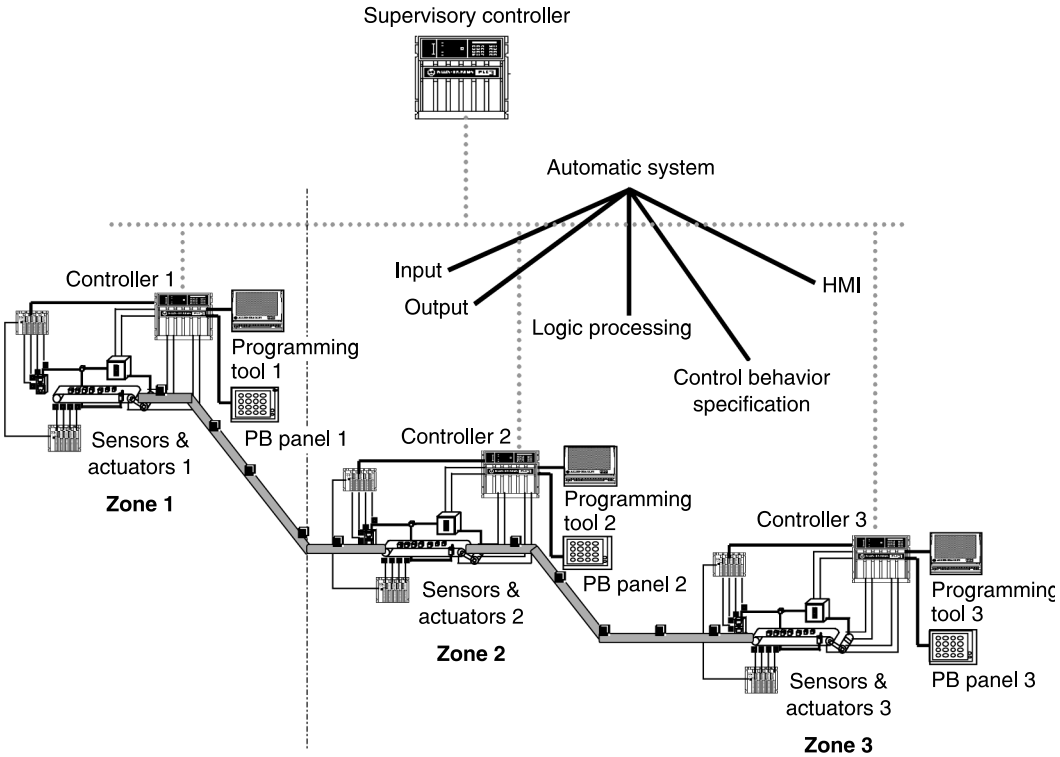


Figure 15.1. Major aspects of an automation system.

the system. In this mode, small disruptions and delays in message delivery could be tolerated. However, when there is a disruption (in part, process, or plant), large bursts of data will be exchanged between a large number of nodes across the system. Under such conditions, disruptions and delays significantly impair system capabilities.

Because of both the large capital investments required and the changing market place, these systems are designed to be flexible with respect to part, process, and plant. Automation systems operate in a periodic manner. The lack of tools to evaluate performance makes it difficult to evaluate the system's performance and vulnerability. These systems are highly engineered and well documented in the design and implementation stages. Demands for reconfigurable architectures and new properties, such as self-organization, require a migration away from current hierarchical structures to loosely coupled networks of devices and subsystems. Control behavior is specified using a special graphical language called *Ladder*, and typical systems offer support for both online and offline program editing.

15.2.2 Operational Problems

Hierarchical architecture and demands for backward compatibility create a plethora of addressing and configuration problems. Cumbersome, expensive, implementations and high commissioning costs are a consequence of such configuration problems. Figure 15.2 shows a typical configuration of input and output (IO) points connected to controllers.

Unique addresses must be assigned to every IO point and single structural component in Figure 15.2. These addresses are typically related to the jumper settings on the device or chassis; thus, a large number addresses are related and structured by manual naming conventions. In addition, depending on the settings on the device or chassis, several items in software packages must also be configured manually.

© 2005 by Chapman & Hall/CRC

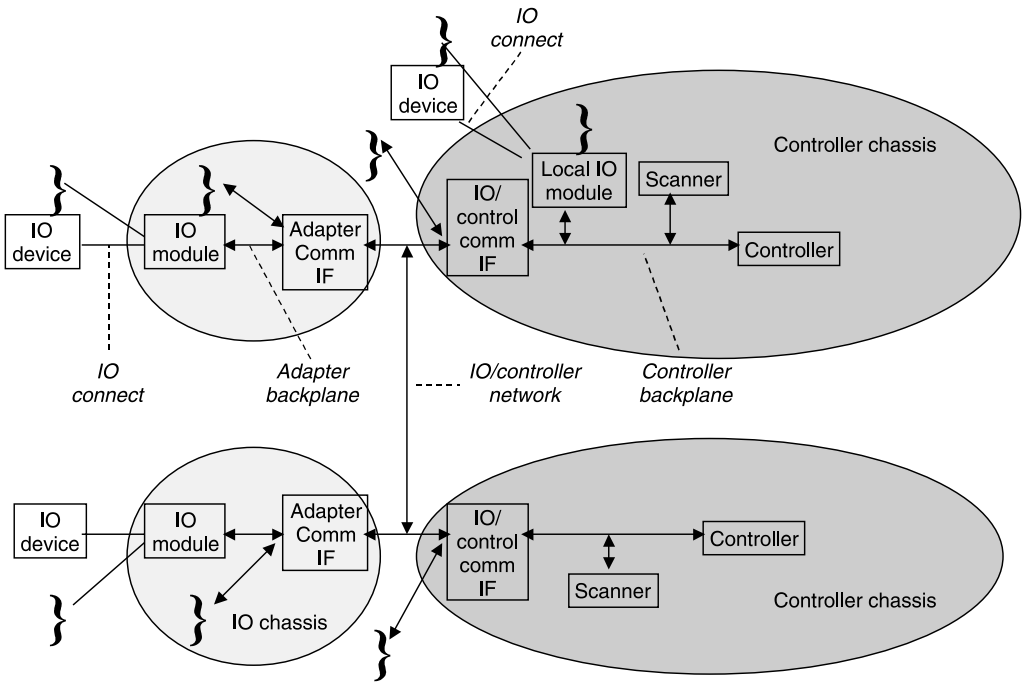


Figure 15.2. Connecting input–output points to controllers.

Naturally, such a landscape of addresses leads to configuration problems. State-of-the-art implementations permit specification of a global set of tags and maintaining multiple links behind the scenes — while this simplifies the user's chores, the underlying problems remain. The current methods of addressing and configuration will not extend to future systems that are characterized by large-scale, complex interaction patterns, and emergent behaviors.

Current methods for commissioning and fault recovery are guided by experience and based on trial-and-error. User-interface devices display localized, controller-centric data and do not support holistic, system-wide decision-making. Predictive nondeterministic models do not accurately represent system dynamics, and hence approaches based on such models have met with limited success. The state-of-practice is a template-based approach that is encoded into Ladder programs. These templates recognize a few commonly occurring errors.

Despite these operational problems, automation systems are a benchmark for safe, predictable, and maintainable systems. HCI devices are robust and reliable. Methods for safe interaction, such as the use of dual palm switches, operator-level de-bouncing, safety mats, etc., are important elements of any safe system. Mechanisms that are used to monitor, track, and study trends in automation systems are good models for such tasks in general distributed systems.

15.2.3 Benefits of Data Fusion

Data fusion can alleviate current operational problems and support development of new architectures that preserve the system characteristics. For example, data fusion techniques based on the foundations discussed in this chapter can provide a more systematic approach to commissioning, fault management (detection, isolation, reporting, and recovery), programming, and security management. Data fusion techniques can be embodied in distributed services that are appropriately located in the system, and such services support a SmartSpace in decision making [9].

© 2005 by Chapman & Hall/CRC

15.3 Data Fusion Foundations

The principle issue for data fusion is to manage uncertainty. We accomplish this task through a goal-seeking paradigm. The application of the goal-seeking paradigm in the context of a multi-level system, such as an automation system, is simplified by a conceptual framework.

15.3.1 Representing System Structure

A conceptual framework is an experience-based stratification of the system, as shown in Figure 15.3. We see a natural organization of the system into levels of a node, station, line, and plant. At each level, the dominant considerations are depicted on the right. There are certain crosscutting abstractions that do not fit well into such a hierarchical organization. For example, the neighborhood of a node admits nodes that are not necessarily in the same station and are accessible. For example, there may be a communications node that has low load that could perform data fusion tasks for a duration when another node in the neighborhood is managing a disruptive fault. Similarly, energy resources in the environment affect all levels of the system.

A conceptual framework goes beyond a simple layering approach or a hierarchical organization. The strata of a conceptual framework are not necessarily organized in a hierarchy. The strata do not provide a *service abstraction* to other strata, like a layer in a software system. Instead, each stratum imposes performance requirements for other strata that it is related to. At runtime, each stratum is responsible for monitoring its own performance based on sensed data. As long as the monitored performance is within tolerance limits specified in the goal-seeking paradigm, the system continues to perform as expected.

15.3.2 Representing System Behavior

We represent the behavior of the system using the goal-seeking paradigm. We briefly review the fundamental state transition paradigm and certain problems associated with this paradigm before discussing the goal-seeking paradigm.

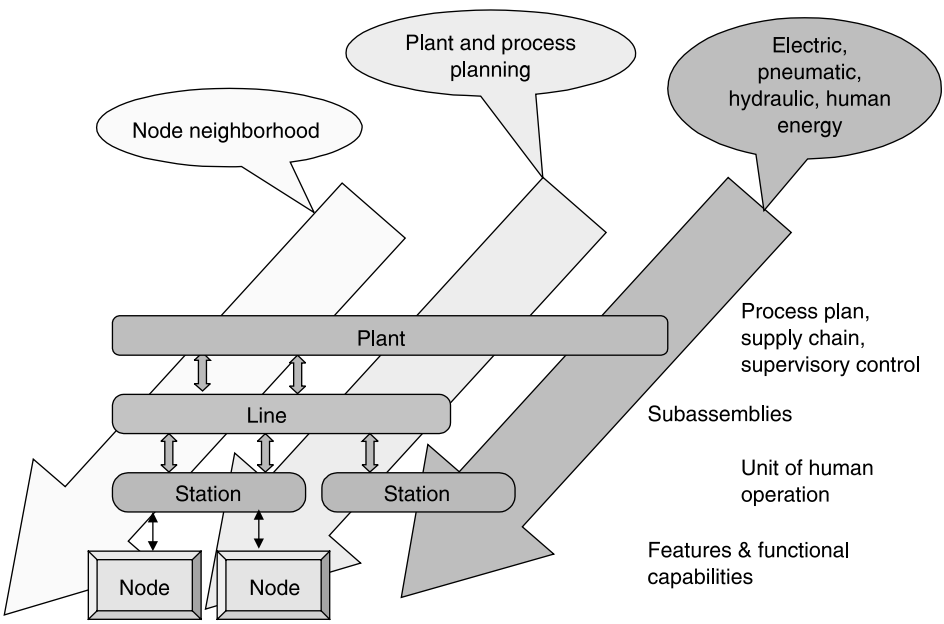


Figure 15.3. Conceptual framework for an automation system.

© 2005 by Chapman & Hall/CRC

15.3.2.1 State Transition Paradigm

The state-transition paradigm is an approach to modeling and describing systems that is based on two key assumptions: first, that states of a system are precisely describable; and second, that the dynamics of the system are also fully describable in terms of states, transitions, inputs that initiate transitions, and outputs that are produced in states.

A state-transition function S_1: $Z_{t1} \otimes X_{t1,t2} \rightarrow Z_{t2}$ defines the behavior of the system by mapping inputs to a new state. For each state, an output function S_2: $Z_{ti} \rightarrow \psi_{ti}$, where $X_{t1,t2}$ is the set of inputs presented to the system in the time interval between $t1$ and $t2$; ψ_i is the set of outputs produced at time ti; Z_{t1} and Z_{t2} are states of the automation system at times $t1$ and $t2$, respectively; the symbol $\otimes$, the Cartesian product, indicates that the variables before the arrow (i.e. inputs $X_{t1,t2}$ and Z_{t1}) are causes for change in the variables after the arrow (i.e. outputs Z_{t2}).

In order to understand such a system, one needs to have complete data on Z_{t1} and $X_{t1,t2}$ and knowledge about S_1 and S_2. This paradigm assumes that only lack of data and knowledge prevent us from completely predicting the future behavior of a system. There is no room for uncertainty or indeterminism. Such a paradigm (sometimes called an IO or stimulus–response paradigm) can be useful in certain limited circumstances for representing the interaction between two systems; and it can be erroneous if it is overextended. There is no consistent or uniform definition of a state, and contextual information that is based on collections of states is ignored. In such circumstances, the specifications in a state-transition paradigm are limited by what is expressed and depend heavily on environmental influences that are received as inputs. While it appears that the state-transition paradigm is simple, natural, and easy to describe, such a formulation can be misleading, especially if the true nature of the system is goal seeking.

15.3.2.2 Goal-Seeking Paradigm

A goal-seeking paradigm is an approach to modeling and describing systems that explicitly supports uncertainty management by using additional artifacts and transformations discussed in the following paragraphs.

The system can choose actions from a range of *alternate actions*, Π, in response to events occurring or expected to occur. These actions represent a choice of decisions that can be made in response to a given or emerging situation.

There is a range of *uncertainties*, Δ, that impact on the success of selected decisions. Uncertainties arise from two sources: first, from an inability to anticipate inputs correctly, either from the automation system or from users; and second, from an incomplete or inaccurate view of the outcome of a decision, even if the input is correctly anticipated. For example, an operator may switch the mode of the system from automatic to manual by mistake, because of malicious intent, or because of poor training. Assuming that the user made appropriate choices, the outcome of a decision can still be uncertain because a component or subsystem of the automation system may fail just before executing the decision.

The range of *consequences*, Ψ, represent outcomes that result from an implementation of system decisions. Consequences are usually outputs that are produced by the system. Some of these outputs may be consumed by users to resolve uncertainties further, and other outputs may actuate devices in the automation system.

The system maintains a *reflection*, Ξ: $\Pi \times \Delta \rightarrow \Psi$, which is its view of the environment. Suppose that the system makes a decision $\pi \in \Pi$, the system benefits from an understanding of what consequence, $\pi \in \Pi$, ψ produces. The consequence ψ is presented as an output, either to humans within a SmartSpace or to the automation system. ψ does not obviously follow ψ as specified by S_2 because of uncertainties in the system.

The *evaluation set*, Λ, represents a *performance scale* that is used to compare the results of alternate actions. That is, suppose the system could make two decisions $\pi_1 \in \Pi$ or $\pi_2 \in \Pi$, and these decisions have consequences $\psi_1, \psi_2 \in \Psi$ respectively; Λ helps to determine which of the two decisions is preferable.

© 2005 by Chapman & Hall/CRC

An *evaluation mapping*, $\Omega: \Psi \times \Pi \rightarrow \Lambda$, is used to compare outcomes of decisions using the performance scale. Λ is specified by taking into account the *extent* or *cost of the effort* associated with a decision, i.e. $\pi \in \Pi$. For any $\pi \in \Pi$ and $\pi \in \Pi$, Ω assigns a value $\lambda \in \Lambda$ and helps to determine the system's preference for a decision–consequence pair (π, ψ).

A *tolerance function*, $\Gamma: \Pi \times \Delta \rightarrow \Lambda$, indicates the degree of satisfaction with the outcome if a given uncertainty $\delta \in \Delta$ comes to pass. For example, if the conditions are full of certainty, then the best (i.e. optimal) decision can be identified. If, however, there are several events that are anticipated (i.e. $|\Delta| \gg 1$), then the performance of the system, as evaluated by Ω, can be allowed to deteriorate for some $\delta \in \Delta$, but this performance must stay within a tolerance limit that will ensure *survival* of the system.

Based on the above artifacts and transformations, the functioning of a system, in a goal-seeking paradigm, is defined as: *Find a decision* $\pi \in \Pi$ *so that the outcome is acceptable (e.g. within tolerance limits) for any possible occurrence of uncertainty* $\delta \in \Delta$, *i.e.* $\Omega(\Xi(\pi, \delta), \pi) \leq \Gamma(\delta, \pi), \forall \delta \in \Delta$.

15.4 Security Management for Discrete Automation

Security management in a discrete automation system is critical because of the current trends towards using open communication infrastructures. Automation systems present challenges to traditional distributed systems, to emerging sensor networks, and for the need to protect the high investments in contemporary assembly lines and factories. Formulating the security management task in the state-transition paradigm is a formidable task and perhaps may never be accomplished because of the scale and the uncertainties that are present. We demonstrate how the goal-seeking formulation helps and the specific data fusion tasks that facilitate security management.

A future automation system is likely to comprise large sensor–actuator networks as important subsystems [8]. Nodes in such a system are resource constrained. Determinism and low jitter are extremely important design considerations Since a node typically contains a simple processor without any controller hierarchy or real-time operating system, it is infeasible to implement a fully secure communications channel for all links without compromising determinism and performance. Further, because of the large installed base of such systems, security mechanisms must be designed to mask the uneven conditioning of the environment [10]. The goal-seeking formulation presented here, and the cooperative data fusion tasks associated, supports such an implementation.

15.4.1 Goal-Seeking Formulation

This section details the artifacts and transformations necessary for security management.

15.4.1.1 Alternate Actions

The set of actions includes options for security mechanisms that are available to a future automation system. Asymmetric cryptography may be applied to either a specific link, a broadcast, a connection, or a session. Digital signatures or certificates may be required or suggested. Frequency hopping may be used to make it difficult for malicious intruders to masquerade or eavesdrop messages. Block ciphering, digest function, or μ-tesla can be applied. These possible actions comprise the set of alternate actions. At each time step, the automation system selects one or more of these alternate actions to maintain security of the system. Because the nodes are resource constrained, it is not possible to implement full encryption for each of the links. Thus, to make the system secure, one or more mechanisms must be selected in a systematic manner depending on the current conditions in the system.

15.4.1.2 Uncertainties

As already discussed in Section 15.3.2.2, there are two sources of uncertainty. First, when a specific user choice is expected as input (a subjective decision), the system cannot guess the choice. Second, component or subsystem malfunctions cannot be predicted. For example, an established channel, connection, packet, or session may be lost. A node receiving a request may be unable to respond to a

© 2005 by Chapman & Hall/CRC

query without compromising local determinism or jitter. The node, channel, or subsystem may be under a denial-of-service attack. There may be a malicious eavesdropper listening to the messages or someone may be masquerading production data to mislead management.

15.4.1.3 Consequences

It is not possible to predict either the occurrence or the time of occurrence of an uncertainty (if it occurs). However, the actions selected may not lead to the consequences intended if one or more uncertainties come to pass. Some example consequences are the communications channel is highly secure at the expected speeds, the channel may be secure at a lower speed, or the channel may be able to deliver data at desired speed without any security. The authentication supplied, digital signature or certificate, is either verified or not verified.

15.4.1.4 Reflection

This transformation maps every decision–uncertainty pair in to a consequence that is presented as an output. The reflection includes all possible choices, without any judgment about either the cost of effort or the feasibility of the consequence in the current system environment.

15.4.1.5 Evaluation Set

The three parameters of interest are security of channel, freshness of data, and authenticity of data. Assuming a binary range for each of these parameters, we get the following scale to evaluate consequences:

1. Highly secure channel with strongly fresh, truly authenticated data.
2. Highly secure channel with strongly fresh, weakly authenticated data.
3. Highly secure channel with weakly fresh, truly authenticated data.
4. Highly secure channel with weakly fresh, weakly authenticated data.
5. Weakly secure channel with strongly fresh, truly authenticated data.
6. Weakly secure channel with strongly fresh, weakly authenticated data.
7. Weakly secure channel with weakly fresh, truly authenticated data.
8. Weakly secure channel with weakly fresh, weakly authenticated data.
9. Total communication failure, no data sent.

15.4.1.6 Evaluation Mapping

Ideally, consequences follow from decisions directly. Because of uncertainties, the consequence obtained may be more desirable or less desirable, depending on the circumstances. The evaluation mapping produces such an assessment for every consequence–decision pair.

15.4.1.7 Tolerance Function

The tolerance function establishes a minimum performance level on the evaluation set that can be used to decide whether or not a decision is acceptable. In an automation system, the tolerance limit typically changes when the system conditions are different. For example, a highly secure channel with strongly fresh, truly authenticated data may be desirable when a user is trying to reprogram a controller, and a weakly secure channel with weakly fresh, weakly authenticated data may be adequate during commissioning phases. The tolerance function can be defined to include such considerations as operating mode and system conditions, with the view of evaluating a decision in the current context of a system.

15.4.2 Applying Data Fusion

Only a few uncertainties may come to pass when a particular alternate action is selected by the system. Hence, we first build a list of uncertainties that apply to every alternate action $\{\Delta_1, \Delta_2, \ldots, \Delta_{|\Pi|}\}$.

© 2005 by Chapman & Hall/CRC

For an action π_k, if the size of the corresponding set of uncertainties $|\Delta_k| > 1$, then data fusion must be applied.

To manage security in an automation system, we need to work with two kinds of sensor. The first kind of sensor is ones that is used to support the automation system, such as to sense the presence of a part, completion of a traversal, or failure of an operation. In addition, there are sensors that help resolve uncertainties. Some of these redundant sensors could either be additional sensors that sense a different modality or another sensor whose value is used in another context to synthesize new information. Some uncertainties can only be inferred as the absence of certain values. For every uncertainty, the set of sensor values (or lack thereof) and inferences are recorded *a priori*. Sensors that must be queried for fresh data are also recorded *a priori*.

For each uncertainty, we also record the stratum in the conceptual framework that dominates the value of an uncertainty. For example, suppose there is an uncertainty regarding the mode of a station. The resolution of this uncertainty is based on whether there is a fault at the station or not. Suppose there is a fault at the station, then the mode at the station must dominate the mode at the line to which the station belongs. Similarly, when the fault has been cleared, the mode of the line must dominate, if the necessary safety conditions are met. Thus, the conceptual framework is useful for resolving uncertainties. The specific implementation of data fusion mechanisms will depend on the capabilities of the nodes. In a resource-constrained environment, we expect such techniques to be integrated with the communications protocols.

15.5 Conclusions

In this chapter we provide a new foundation for data fusion based on two concepts: a conceptual framework and the goal-seeking paradigm. The conceptual framework emphasizes the dominant structures in the system. The goal-seeking paradigm is a mechanism to represent system evolution that explicitly manages uncertainty. The goal-seeking formulation for data fusion helps to distinguish between subjective decisions that resolve uncertainty by involving humans and objective decisions that can be executed by computers. These notions are useful for critical tasks, such as security management in large-scale distributed systems. Investigations in this area, and further refinement of the goal-seeking formulation for instrumentation and measurement applications, are likely to lead to future systems that facilitate holistic user decision-making.

Acknowledgements

This work is supported in part by a University of Akron, College of Engineering Research Startup Grant, 2002–2004, to Dr. Sastry and an NSF Grant # IIS-0239914 to Professor Iyengar.

References

[1] Brooks, R.R. and Iyengar, S.S., *Multi-Sensor Fusion: Fundamentals and Applications with Software*, Prentice Hall, NJ, 1997.

[2] Iyengar, S.S. *et al.*, Distributed sensor networks for real-time systems with adaptive configuration, *Journal of the Franklin Institute*, 338, 571, 2001.

[3] Kannan, R. *et al.*, Sensor-centric quality of routing in sensor networks, in *Proceedings of IEEE Infocom*, April 2003.

[4] Akyildiz, I.F. *et al.*, Wireless sensor networks: a survey, *Computer Networks*, 38, 393, 2002.

[5] Agre, J. *et al.*, A taxonomy for distributed real-time control systems, *Advances in Computers*, 49, 303, 1999.

[6] Slansky, D., Collaborative discrete automation systems define the factory of the future, ARC Strategy Report, May 2003.

[7] Mesarovic, M.D. and Takahara, Y., *Mathematical Theory of General Systems*, Academic Press, 1974.

© 2005 by Chapman & Hall/CRC

[8] Sastry, S., and Iyengar, S.S., Sensor technologies for future automation systems, *Sensor Letters*, 2(1), 9–17, 2004.
[9] Sastry, S., A smartspace for automation, *Assembly Automation*, 24(2), 201–209, 2004.
[10] Sathyanarayanan, M., Pervasive computing: vision and challenges, *Pervasive Computing*, (August), 10, 2001.

© 2005 by Chapman & Hall/CRC

16

Measurement-Based Statistical Fusion Methods For Distributed Sensor Networks

Nageswara S.V. Rao

16.1 Introduction

In distributed sensor networks (DSNs), fusion problems naturally arise when overlapping regions are covered by a set of sensor nodes. The sensor nodes typically consist of specialized sensor hardware and/or software, and consequently their outputs are related to the actual object features in a complicated manner, which is often modeled by probability distributions. While fusion problems have been solved for centuries in various disciplines, such as political economy, the specific nature of fusion problems of DSNs require nonclassical approaches. Early information fusion methods required statistical independence of sensor errors, which greatly simplified the fuser design; for example, a weighted majority rule suffices in detection problems. Such a solution is not applicable to DSNs since the sensors could be highly correlated while sensing common regions or objects, and thereby violate the statistical independence property. Another classical approach to fuser design relies on the Bayesian method that minimizes a suitable expected risk. A practical implementation of this method requires closed-form analytical expressions for sensor distributions to generate efficiently computable fusers. Several popular distributed decision fusion methods belong to this class [1]. In DSNs, the sensor distributions can be arbitrarily complicated. In addition, deriving closed-form expressions for sensor distributions is a very difficult and expensive task, since it requires knowledge of a variety of areas, such as device physics, electrical engineering, and statistical modeling. Furthermore, the problem of selecting a fuser from a carefully chosen function class is easier in an information-theoretic sense, than inferring a completely unknown distribution [2].

© 2005 by Chapman & Hall/CRC

In operational DSNs, it is quite practical to collect "data" by sensing objects and environments with known parameters. Thus, fusion methods that utilize empirical data available from the observation and/or experimentation will be of practical value. In this chapter, we present an overview of rigorous methods for fusion rule estimation from empirical data, based on empirical process theory and computational learning theory. Our main focus is on methods that provide *performance guarantees based on finite samples* from a statistical perspective. We do not cover *ad hoc* fusion rules with no performance bounds or results based on asymptotic guarantees valid *only* as the sample size approaches infinity. This approach is based on a statistical formulation of the fusion problem, and may not fully capture the nonstatistical aspects, such as calibration and registration. These results, however, provide an analytical justification for sample-based approaches to a very general formulation of the sensor fusion problem.

The organization of this chapter is as follows. We briefly describe the classical sensor fusion methods from a number of disciplines in Section 16.2. We present the formulation of a generic sensor fusion problem in Section 16.3. In Section 16.4 we present two solutions based on the empirical risk minimization methods using neural networks and vector space methods. In Section 16.5 we present solutions based on the Nadaraya–Watson statistical estimator. We describe applications of these methods in Section 16.6. In Section 16.7 we address the issues of relative performance of the fused system compared with the component sensors. We briefly discuss metafusers that combine individual fusers in Section 16.8.

16.2 Classical Fusion Problems

Historically, the information fusion problems predate DSNs by a few centuries. Fusion of information from multiple sources to achieve performances exceeding those of individual sources has been recognized in diverse areas such as political economy models [3] and composite methods [4]; a brief overview of these works can be found in [5]. The fusion methods continued to be applied in a wide spectrum of areas, such as reliability [6], forecasting [7], pattern recognition [8], neural networks [9], decision fusion [1,10], and statistical estimation [11].

If the sensor error distributions are known, the several fusion rule estimation problems have been solved typically by methods that do not require samples. Earlier work in pattern recognition is due to Chow [8], who showed that a weighted majority fuser is optimal in combining outputs from pattern recognizers under statistical independence conditions. Furthermore, the weights of the majority fuser can be derived in closed form in terms of the individual detection probabilities of patten recognizers. A simpler version of this problem has been studied extensively in political economy models (for example, see [3] for an overview). Under the Condorcet jury model of 1786, the simple majority rule has been studied in combining the 1–0 probabilistic decisions of a group of N statistically independent members. If each member has probability p of making a correct decision, then the probability that the majority makes the correct decision is

$$p_N = \sum_{i=N/2}^{N} \binom{N}{i} p^i (1-p)^{N-i}$$

Then we have an interesting dichotomy: (a) if $p > 0.5$, then $p_N > p$ and $p_N \to 1$ as $N \to \infty$; and (b) if $p < 0.5$, then $p_N < p$ and $p_N \to 0$ as $N \to \infty$. For the boundary case $p = 0.5$ we have $p_N = 0.5$. Interestingly, this result was rediscovered by von Neumann in 1959 in building reliable computing devices using unreliable components by taking a majority vote of duplicated components.

The distributed detection problem [1], studied extensively in the target tracking area, can be viewed as a generalization of the above two problems. The Boolean decisions from a system of detectors are combined by minimizing a suitably formulated Bayesian risk function. The risk function is derived from the densities of detectors, and the minimization is typically carried out using analytical or

© 2005 by Chapman & Hall/CRC

deterministic optimization methods. A special case of this problem is very similar to [8], where the risk function corresponds to the probability of misclassification and its minima are achieved by a weighted majority rule. Another important special case is the correlation coefficient method [12] that explicitly accounts for the correlations between the subsets of detectors in designing the fuser. In these studies, the sensor distributions are assumed to be known, which is quite reasonable in the areas these methods are applied. While several of these solutions can be converted into sample-based ones [13,14], these are not designed with measurements as primary focus; furthermore, they address only special cases of the generic sensor fusion problem.

16.3 Generic Sensor Fusion Problem

We consider a *generic sensor system* of N sensors, where the sensor S_i, $i = 1, 2, \ldots, N$, outputs $Y^{(i)} \in \Re^d$ corresponding to input $X \in \Re^d$ according to the distribution $P_{Y^{(i)}|X}$. Informally, the input X is the quantity that needs to be "estimated" or "measured" by the sensors, such as the presence of a target or a value of the feature vector. We define the *expected error* of the sensor S_i as

$$I(S_i) = \int C\left(X, Y^{(i)}\right) \mathrm{d}P_{Y^{(i)},X}$$

where $C : \Re^d \times \Re^d \mapsto \Re$ is the cost function. Intuitively $I(S_i)$ is a measure of how good the sensor S_i is in "sensing" the input feature X. If S_i is a detector [1], then we have $X \in \{0, 1\}$ and $Y^{(i)} \in \{0, 1\}$, where $X = 1$ (0) corresponds to the presence (absence) of a target. Then

$$I(S_i) = \int \left[X \oplus Y^{(i)}\right] \mathrm{d}P_{Y^{(i)},X}$$

is the probability of misclassification (false alarm and missed detection) of S_i, where $\oplus$ is the exclusive-OR operation.

The *measurement error* corresponds to the randomness involved in measuring a particular value of the feature X, which is distributed according to $P_{Y^{(i)}|X}$. The *systematic error* at X corresponds to $E[C(X, Y^{(i)})|X]$ which must be zero in the case of a perfect sensor. This error is often referred to as the bias error.

We consider an example of a sensor system of two sensors such that, for the first sensor, we have $Y^{(1)} = a_1 X + Z$, where Z is normally distributed with zero mean, and is independent of X. Thus the first sensor has a constant scaling error and a random additive error. For the second sensor, we have $Y^{(2)} = a_2 X + b_2$, which has a scaling and bias error. Let X be uniformly distributed over $[0, 1]$, and $C[X, Y] = (X - Y)^2$. Then, we have $I(S_1) = (1 - a_1)^2$ and $I(S_2) = (1 - a_2 - b_2)^2$, which are nonzero in general.

We consider a fuser $f : \Re^{Nd} \mapsto \Re^d$ that combines, the outputs of sensors $Y = \left(Y^{(1)}, Y^{(2)}, \ldots, Y^{(N)}\right)$ to produce the fused output $f(Y)$. We define the *expected error* of the fuser f to be

$$I_{\mathrm{F}}(f) = \int C(X, f(Y)) \, \mathrm{d}P_{Y,X}$$

where $Y = \left(Y^{(1)}, Y^{(2)}, \ldots, Y^{(N)}\right)$. The objective of fusion is to achieve low values of $I_{\mathrm{F}}(f)$ for which both systematic and measurement errors must be accounted for. The fuser is typically chosen from a family of fusion rules $\mathcal{F} = \{f : \Re^N \mapsto \Re\}$ which could be either explicitly or implicitly identified. The *expected best* fusion rule $f^\star$ minimizes $I_{\mathrm{F}}(.)$ over $\mathcal{F}$, i.e.

$$I_{\mathrm{F}}(f^\star) = \min_{f \in \mathcal{F}} I_{\mathrm{F}}(f)$$

© 2005 by Chapman & Hall/CRC

For example, $\mathcal{F}$ could be the set of sigmoidal neural networks obtained by varying the weight vector for a fixed architecture. In this case $f^\star = f_{w^\star}$ corresponding to the weight vector $w^\star$ that minimizes $I_F(.)$ over all the weight vectors.

Continuing our example, we consider the fuser

$$f\left(Y^{(1)}, Y^{(2)}\right) = \frac{Y^{(1)}}{2a_1} + \frac{1}{2a_2}(Y^{(2)} - b)$$

For this fuser, we have $I_F(f) = 0$, since the bias b is subtracted from $Y^{(2)}$ and the multipliers cancel the scaling error. In practice, however, such a fuser can be designed only with a significant insight into sensors, in particular with a detailed knowledge of the distributions.

In this formulation, $I_F(.)$ depends on the error distribution $P_{Y,X}$, and hence $f^\star$ cannot be computed even in principle if the former is not known. We consider that only an independently and identically distributed (iid) l-sample

$$(X_1, Y_1), (X_2, Y_2), \ldots, (X_l, Y_l)$$

is given, where $Y_i = (Y_i^{(1)}, Y_i^{(2)}, \ldots, Y_i^{(N)})$ and $Y_i^{(j)}$ is the output of S_j in response to input X_i. We consider an estimator $\hat{f}$, based *only* on a sufficiently large sample, such that

$$P_{Y,X}^l\left[I_F(\hat{f}) - I_F(f^\star) > \epsilon\right] < \delta \tag{16.1}$$

where $\epsilon > 0$ and $0 < \delta < 1$, and $P_{Y,X}^l$ is the distribution of iid l-samples. For simplicity, we subsequently denote $P_{Y,X}^l$ by P. This condition states that the "error" of $\hat{f}$ is within ϵ of optimal error (of $f^\star$) with an arbitrary high probability $1 - \delta$, *irrespective* of the underlying sensor distributions. It is a reasonable criterion, since $\hat{f}$ is to be "chosen" from an infinite set, namely $\mathcal{F}$, based only on a finite sample. The conditions that are strictly stronger than Equation (16.1) are generally not possible. To illustrate this, consider the condition $P_{Y,X}^l[I_F(\hat{f}) > \epsilon] < \delta$ for the case $\mathcal{F} = \{f : [0,1]^N \mapsto \{0,1\}\}$. This condition cannot be satisfied, since for any $f \in \mathcal{F}$ there exists a distribution for which $I_F(f) > 1/2 - \rho$ for any $\rho \in [0,1]$; see Theorem 7.1 of [15] for details.

To illustrate the effects of finite samples in the above example, consider that we generate three values for X given by $\{0.1, 0.5, 0.9\}$ with corresponding Z values given by $\{0.1, -0.1, -0.3\}$. The corresponding values for $Y^{(1)}$ and $Y^{(2)}$ are given by $\{0.1a_1 + 0.1, 0.5a_1 - 0.1, 0.9a_1 - 0.3\}$ and $\{0.1a_2 + b_2, 0.5a_2 + b_2, 0.9a_1 + b_2\}$ respectively. Consider the class of linear fusers such that $f\left(Y^{(1)}, Y^{(2)}\right) = w_1 Y^{(1)} + w_2 Y^{(2)} + w_3$. Based on the measurements, the following weights enable the fuser outputs to exactly match X values for each of the measurements:

$$w_1 = \frac{1}{0.2 - 0.4a_1}, \quad w_2 = \frac{1}{0.4a_2}, \quad w_3 = \frac{0.1a_1 + 0.1}{0.4a_1 + 0.1} - \frac{0.1a_2 + b_2}{0.4a_2}$$

While the fuser with these weights achieves zero error on the measurements it does not achieve zero value for I_F. Note that a fuser with zero expected error exists, and can be computed if the sensor distributions are given. The idea behind the criterion in Equation (16.1) is to achieve performances close to an optimal fuser using only a sample. To meet this criterion one needs to select a suitable $\mathcal{F}$, and then achieve small error on a sufficiently large sample, as will be illustrated subsequently.

The generic sensor fusion problem formulated here is fairly general and requires very little information about the sensors. In the context of a DSN, each sensor could correspond to a node consisting of a hardware device, a software module, or a combination. We describe some concrete examples in Section 16.6.

© 2005 by Chapman & Hall/CRC

16.3.1 Related Formulations

Owing to the generic nature of the sensor fusion problem described here, it is related to a number of similar problems in a wide variety of areas. Here, we briefly show its relationship to some of the well-known methods in engineering areas. If the sensor error distributions are known, then several fusion rule estimation problems have been solved by methods not requiring the samples. The distributed detection problem based on probabilistic formulations has been extensively studied [1]. These problems are special cases of the generic fusion problem such that $X \in \{0, 1\}$ and $Y \in \{0, 1\}^N$, but the difference is that they assume that various probabilities are available. If only measurements are available then these methods are not applicable. While solutions to the generic sensor fusion problem are applicable here, much tighter performance bounds are possible since distribution detection is a special (namely Boolean) case of the generic sensor fusion problem [16,17]. Also, in many cases the solutions based on known distributions cases can be converted to sample-based ones [13].

Many of the existing information integration techniques are based on maximizing *a posteriori* probabilities of hypotheses under a suitable probabilistic model. However, in situations where the probability densities are unknown (or difficult to estimate) such methods are ineffective. One alternative is to estimate the density based on a sample. But, as illustrated in general by [2], the density estimation is more difficult than the subsequent problem of estimating a function chosen from a family with bounded capacity. This property holds for several pattern recognition and regression estimation problems [2]. In the context of feedforward neural networks that "learn" a function based on a sample, the problem is to identify the weights of a network of chosen architecture. The choice of weights corresponds to a particular network $\hat{f}$ from a family $\mathcal{F}$ of neural networks of a particular architecture. This family $\mathcal{F}$ satisfies bounded capacity [18] and Lipshitz property [19]. Both these properties are conducive to the statistical estimation of $\hat{f}$ as explained in the next section. On the other hand, no such information is available about the class from which the unknown density is chosen, which makes it difficult to estimate the density.

It is not necessary that all sensor distributions be unknown to apply our formulation. Consider that the joint conditional distribution $P_{Y^{(1)},\ldots,Y^{(M)}|Y^{(M+1)},\ldots,Y^{(N)}}$ of the sensors $S_1, \ldots, S_M$, for $M < N$, is known. Then we can rewrite

$$I_{\mathrm{F}}(f) = \int \Phi(X, f(Y))\, \mathrm{d}P_{Y^{(M+1)},\ldots,Y^{(N)},X}$$

where $\Phi(.)$ is suitably derived from the original cost function $C(.)$ and the known part of the conditional distribution. Then, the solutions to the generic sensor fusion problem can be applied to this new cost function with only a minor modification. Since the number of variables with unknown distribution is now reduced, the statistical estimation process is easier. It is important to note that it is not sufficient to know the individual distributions of the sensors, but the *joint* conditional distributions are required to apply this decomposition. In the special case of statistical independence of sensors the joint distribution is just the product, which makes the transformation easier. In general for sensor fusion problems, however, the interdependence between the sensors is a main feature to be exploited to overcome the limitations of single sensors.

16.4 Empirical Risk Minimization

In this section we present fusion solutions based on the empirical risk minimization methods [2]. Consider that the empirical estimate

$$I_{\mathrm{emp}}(f) = \frac{1}{l}\sum_{i=1}^{l}\left[X_i - f\left(Y_i^{(1)}, Y_i^{(2)}, \ldots, Y_i^{(N)}\right)\right]^2$$

© 2005 by Chapman & Hall/CRC

is minimized by $\hat{f} \in \mathcal{F}$. Using Vapnik's empirical risk minimization method [2], for example, we can show [20] that if $\mathcal{F}$ has finite capacity, then under bounded error, or bounded relative error for sufficiently large sample

$$P_{Y,X}^{l}\left[I_{\mathrm{F}}(\hat{f}) - I_{\mathrm{F}}(f^{\star}) > \epsilon\right] < \delta$$

for arbitrarily specified $\epsilon > 0$ and δ, $0 < \delta < 1$. Typically, the required sample size is expressed in terms of ϵ and δ and the parameters of $\mathcal{F}$. The most general result that ensures this condition is based on the scale-sensitive dimension [21]. This result establishes the basic tractability of the sensor fusion problem, but often results in very loose bounds for the sample size. By utilizing specific properties of $\mathcal{F}$, tighter sample size estimates are possible. In this section we describe two such classes of $\mathcal{F}$ and their sample size estimators.

16.4.1 Feedforward Sigmoidal Networks

We consider a feedforward network with a single hidden layer of k nodes and a single output node. The output of the jth hidden node is $\sigma(b_j^{\mathrm{T}} y + t_j)$, where $y \in [-B, B]^N$, $b_j \in \Re^N$, $t_j \in \Re$, and the nondecreasing $\sigma : \Re \mapsto [-1, +1]$ is called the *activation function*. The output of the network corresponding to input y is given by

$$f_w(y) = \sum_{j=1}^{k} a_j \sigma(b_j^{\mathrm{T}} y + t_j)$$

where $w = (w_1, w_2, \ldots, w_{k(d+2)})$ is the *weight vector* of the network consisting of $a_1, a_2, \ldots, a_k$, $b_{11}, b_{12}, \ldots, b_{1d}, \ldots, b_{k1}, \ldots, b_{kd}$, and $t_1, t_2, \ldots, t_k$. Let the set of *sigmoidal feedforward networks* with *bounded weights* be denoted by

$$\mathcal{F}_{\mathcal{W}}^{\gamma} = \{f_w : w \in [-W, W]^{k(d+2)}\}$$

where $0 < \gamma < \infty$, and $\sigma(z) = \tanh^{-1}(\gamma z)$, $0 < W < \infty$.

The following theorem provides several sample size estimates for the fusion rule estimation problem based on the different properties of neural networks.

Theorem 16.1 [22] *Consider the class of feedforward neural networks $\mathcal{F}_{\mathcal{W}}^{\gamma}$. Let $X \in [-A, A]$ and $R = 8(A + kW)^2$. Given a sample of size at least*

$$\frac{16R}{\epsilon^2}\left\{\ln(18/\delta) + 2\ln(8R/\epsilon^2) + \ln(2\gamma^2 W^2 kR/\epsilon) + \frac{\gamma W^2 kR}{\epsilon}\left[\left(\frac{\gamma W^2 kR}{\epsilon} - 1\right)^{N-1} + 1\right]\right\}$$

the empirical best neural network $\hat{f}_w$ in $\mathcal{F}_{\mathcal{W}}^{\gamma}$ approximates the expected best $f^{\star}{}_w$ in $\mathcal{F}_{\mathcal{W}}^{\gamma}$ such that

$$P\left[I_{\mathrm{F}}(\hat{f}_w) - I_{\mathrm{F}}(f^{\star}{}_w) > \epsilon\right] < \delta$$

© 2005 by Chapman & Hall/CRC

The same condition can also be ensured under the sample size

$$\frac{16R}{\epsilon^2}\left(\ln(18/\delta) + 2\ln(8R/\epsilon^2) + k(d+2)\ln(L_w R/\epsilon)\right)$$

where $L_w = \max(1, WB\gamma^2/4, W\gamma^2/4)$, *or, for* $\gamma = 1$,

$$\frac{128R}{\epsilon^2}\max\left\{\ln\left(\frac{8}{\delta}\right), \ln\left(\frac{16e(k+1)R}{\epsilon}\right)\right\}$$

These sample sizes are based on three qualitatively different parameters of $\mathcal{F}$, namely (a) Lipschitz property of $f(y) \in \mathcal{F}$ with respect to input y [23], (b) compactness of weight set and smoothness of $f \in \mathcal{F}$ with respect to weights, and (c) VC-dimension of translates of sigmoid units [24]. The three sample estimates provide three different means for controlling the sample size depending on the available information and intrinsic characteristics of the neural network class $\mathcal{F}_{\mathcal{W}}^{\gamma}$. The sample sizes in the first and second bounds can be modified by changing the parameter γ. For example, by choosing $\gamma = \epsilon/(W^2kR)$ the first sample size can be reduced to a simpler form

$$\frac{16R}{\epsilon^2}\left[\ln(18/\delta) + \ln\left(\frac{128R}{W^2k\epsilon^2}\right)\right]$$

Also, by choosing $\gamma^2 = 4/(W_{\max}(1, B))$, we have a simpler form of the second sample size estimate

$$\frac{16R}{\epsilon^2}\left[\ln\left(\frac{1152}{\delta}\right) + k(d+2)\ln(R/\epsilon)\right]$$

for $R \geq 1$. In practice, it could be useful to compute all three bounds and choose the smallest one.

The problem of computing the empirical best neural network $\hat{f}_w$ is NP-complete for very general subclass of $\mathcal{F}_{\mathcal{W}}^{\gamma}$ [25]. In several practical cases very good results have been obtained using the backpropagation algorithm, which provides an approximation to $\hat{f}_w$. For the vector space method in the next section, the computation problem is polynomial-time solvable.

16.4.2 Vector Space Methods

We now consider that $\mathcal{F}$ forms a finite dimensional vector space, and as a result: (a) the sample size is a simple function of the dimensionality of $\mathcal{F}$, (b) $\hat{f}$ can be easily computed by well-known least-squares methods in polynomial time, and (c) no smoothness conditions are required on the functions or distributions.

Theorem 16.2 [20] *Let* f^* *and* $\hat{f}$ *denote the expected best and empirical best fusion functions chosen from a vector space* $\mathcal{F}$ *of dimension* d_V *and range* $[0, 1]$. *Given an iid sample of size*

$$\frac{512}{\epsilon^2}\left[d_V \ln\left(\frac{64e}{\epsilon} + \ln\frac{64e}{\epsilon}\right) + \ln(8/\delta)\right]$$

we have $P[I_F(\hat{f}) - I_F(f^*) > \epsilon] < \delta$.

© 2005 by Chapman & Hall/CRC

If $\{f_1, f_2, \ldots, f_{d_V}\}$ is a basis of $\mathcal{F}$, $f \in \mathcal{F}$ can be written as $f(y) = \sum_{i=1}^{d_V} a_i f_i(y)$ for $a_i \in \Re$. Then consider $\hat{f} = \sum_{i=1}^{d_V} \hat{a}_i f_i(y)$ such that $\hat{a} = (\hat{a}_1, \hat{a}_2, \ldots, \hat{a}_{d_V})$ minimizes the cost expressed as

$$I_{\text{emp}}(a) = \frac{1}{l} \sum_{k=1}^{l} \left(X_k - \sum_{i=1}^{d_V} a_i f_i(Y_k) \right)^2$$

where $a = (a_1, a_2, \ldots, a_{d_V})$. Then $I_{\text{emp}}(a)$ can be written in the quadratic form $a^{\mathrm{T}} C a + a^{\mathrm{T}} D$, where $C = [c_{ij}]$ is a positive definite symmetric matrix, and D is a vector. This form can be minimized in polynomial-time using quadratic programming methods [27].

This method subsumes two very important cases:

1. *Potential Functions.* The potential functions of Aizerman *et al.* [28], where $f_i(y)$ is of the form $\exp[(y-\alpha)^2/\beta]$ for suitably chosen constants α and β, constitute an example of the vector space methods. An incremental algorithm was originally proposed for the computation of the coefficient vector a, for which finite sample results have been derived recently [29] under certain conditions.
2. *Special Neural Networks.* In the two-layer sigmoidal networks of [30], the unknown weights are only in the output layer, which enables us to express each network in the form $\sum_{k=1}^{d_V} a_i \eta_i(y)$ with universal $\eta_i(.)$ values. These networks have been shown to approximate classes of the continuous functions with arbitrarily specified precision in a manner similar to the general single-layer sigmoidal networks, as shown in [31].

16.5 Statistical Estimators

The fusion rule estimation problem is very similar to the regression estimation problem. In this section we present a polynomial-time (in sample size l) computable Nadaraya–Watson estimator which guarantees the criterion in Equation (16.1) under additional smoothness conditions. We first present some preliminaries needed for the main result. Let Q denote the unit cube $[0,1]^N$ and $\mathcal{C}(Q)$ denote the set of all continuous functions defined on Q. The modulus of smoothness of $f \in \mathcal{C}(Q)$ is defined as

$$\omega_\infty(f; r) = \sup_{\|y-z\|_\infty < r,\, y,z \in Q} |f(y) - f(z)|$$

where $\| y - z \|_\infty = \max_{i=1}^{M} |y_i - z_i|$. For $m = 0, 1, \ldots$, let Q_m denote a family of diadic cubes (Haar system) such that $Q = \bigcup_{J \in Q_m} J$, $J \cap J' = \emptyset$ for $J \neq J'$, and the N-dimensional volume of J, denoted by $|J|$, is 2^{-Nm}. Let $1_J(y)$ denote the indicator function of $J \in Q_m$: $1_J(y) = 1$ if $y \in J$, and $1_J(y) = 0$ otherwise. For given m, we define the map P_m on $\mathcal{C}(Q)$ as follows: for $f \in \mathcal{C}(Q)$, we have $P_m(f) = P_m f$ defined by

$$P_m f(y) = \frac{1}{|J|} \int_J f(z)\, \mathrm{d}z$$

for $y \in J$ and $J \in Q_m$ [32]. Note that $P_m f : Q \mapsto [0,1]$ is a discontinuous (in general) function which takes constant values on each $J \in Q_m$. The *Haar kernel* is given by

$$P_m(y,z) = \frac{1}{|J|} \sum_{J \in Q_m} 1_J(y) 1_J(z)$$

for $y, z \in Q$.

© 2005 by Chapman & Hall/CRC

Given l-sample, the Nadaraya–Watson estimator based on Haar kernels is defined by

$$\hat{f}_{m,l}(y) = \frac{\sum_{j=1}^{l} X_j P_m(y, Y_j)}{\sum_{j=1}^{l} P_m(y, Y_j)} = \frac{\sum_{Y_j \in J} X_j}{\sum_{Y_j \in J} 1_J(Y_j)}$$

for $y \in J$ [33,34]. This expression indicates that $\hat{f}_{m,l}(y)$ is the mean of the function values corresponding to Y_j values in J that contains y. The Nadaraya–Watson estimator based on more general kernels is well known in statistics literature [35]. Typical performance results of this estimator are in terms of asymptotic results, and are not particularly targeted toward fast computation. The above computationally efficient version based on Haar kernels is due to [34], which was subsequently shown to yield finite sample guarantees in [36] under the finiteness of capacity of $\mathcal{F}$ in addition to smoothness. This estimator can be used to solve the generic sensor fusion problem.

Theorem 16.3 [37] *Consider a family of functions $\mathcal{F} \subseteq \mathcal{C}(Q)$ with range $[0, 1]$ such that $\omega_\infty(f; r) \leq kr$ for some $0 < k < \infty$. We assume that: (i) there exists a family of densities $\mathcal{P} \subseteq \mathcal{C}(Q)$; (ii) for each $p \in \mathcal{P}$, $\omega_\infty(p; r) \leq kr$; and (iii) there exists $\mu > 0$ such that for each $p \in \mathcal{P}$, $p(y) > \mu$ for all $y \in [0, 1]^N$. Suppose that the sample size, l, is larger than*

$$\frac{2^{2m+4}}{\epsilon_1^2}\left(\left\{\frac{k2^m}{\epsilon_1}\left[\left(\frac{k2^m}{\epsilon_1} - 1\right)^{N-1} + 1\right] + m\right\}\ln\left(2^{m+1}k/\epsilon_1\right) + \ln\left(\frac{2^{2m+6}}{(\delta - \lambda)\epsilon_1^4}\right)\right)$$

where $\epsilon_1 = \epsilon(\mu - \epsilon)/4$, $0 < \beta < N/[2(N+1)]$, $m = \lceil \log l\beta/N \rceil$ and

$$\lambda = b\left(\frac{2}{\epsilon}\right)^{1/N+1-1/2\beta} + b\left(\frac{2}{\epsilon_1}\right)^{1/N+1-1/2\beta}$$

Then for any $f \in \mathcal{F}$, we have $P[I_F(\hat{f}_{m,l}) - I_F(f^*)| > \epsilon] < \delta$.

We note that the value of $\hat{f}_{m,l}(y)$ at a given y is the ratio of local sum of X_i values to the number of Y_i values in J that contains y. The range tree [38] can be constructed to store the cells J that contain at least one Y_i; with each such cell, we store the number of the Y_i values that are contained in J and the sum of the corresponding X_i values. The time complexity of this construction is $O[l(\log l)^{N-1}]$ [38]. Using the range tree, the values of J containing y can be retrieved in $O[(\log l)^N]$ time [36].

The smoothness conditions required in Theorem 16.3 are not very easy to verify in practice. However, this estimator is found to perform well in a number of applications, including those that do not have smoothness properties (see Section 16.6). Several other statistical estimators can also be used for fusion rule estimation, but finite sample results must be derived to ensure the condition in Equation (16.1). Such finite sample results are available for adapted nearest-neighbor rules and regressograms [36] which can also be applied for the fuser estimation problem.

16.6 Applications

We describe three concrete applications to illustrate the performance of methods described in the previous sections — the first two are simulation examples and the third one is an experimental system. In addition, the first two examples also provide results obtained with the nearest neighbor rule, which is analyzed in [23]. In the second example, we also consider another estimate, namely the empirical

© 2005 by Chapman & Hall/CRC

Table 16.1. Fusion of function estimators: mean-square-error over test set

Training set	Testing set	Nadaraya–Watson	Nearest neighbor	Neural network
(a) $d=3$				
100	10	0.000902	0.002430	0.048654
1000	100	0.001955	0.003538	0.049281
10000	1000	0.001948	0.003743	0.050942
(b) $d=5$				
100	10	0.004421	0.014400	0.018042
1000	100	0.002944	0.003737	0.021447
10000	1000	0.001949	0.003490	0.023953

decision rule described in [39]. Pseudo random number generators are used in both the simulation examples.

Example 16.1: *Fusion of Noisy Function Estimators* [37]. Consider five estimators of a function $g : [0,1] \mapsto [0,1]$ such that the ith estimator outputs a corrupted value $Y^{(i)} = g_i(X)$ of $g(X)$ when presented with input $X \in [0,1]^d$. The fused estimate $f(g_1(X), \ldots, g_5(X))$ must closely approximate $g(X)$. Here g is realized by a feedforward neural network, and, for $i = 1, 2, \ldots, 5$, $g_i(X) = g(X)(1/2 + iZ/10)$ where Z is uniformly distributed over $[-1, 1]$. Thus we have $1/2 - i/10 \leq g_i(X)/g(X) \leq 1/2 + i/10$. Table 16.1 corresponds to the mean-square-error in the estimation of f for $d=3$ and $d=5$, respectively, using the Nadaraya–Watson estimator, nearest neighbor rule, and a feedforward neural network with backpropagation learning algorithm. Note the superior performance of the Nadaraya–Watson estimator. □

Example 16.2: *Distributed Detection* [22–39]. We consider five sensors such that $Y \in \{H_0, H_1\}^5$ such that $X \in \{H_0, H_1\}$ corresponds to a "correct" decision, which is generated with equal probabilities, i.e. $P(X = H_0) = P(X = H_1) = 1/2$. The error of sensor S_i, $i = 1, 2, \ldots, 5$, is described as follows: the output $Y^{(i)}$ is a correct decision with probability of $1 - i/10$, and is the opposite with probability $i/10$. The task is to combine the outputs of the sensors to predict the correct decision. The percentage error of the individual detectors and the fused system based on the Nadaraya–Watson estimator is presented in Table 16.2. Note that the fuser is consistently better than the best sensor S_1 beyond sample sizes of the order of 1000. The performance results of the Nadaraya–Watson estimator, empirical decision rule, nearest neighbor rule, and the Bayesian rule based on the analytical formulas are presented in Table 16.3. The Bayesian rule is computed based on the formulas used in the data generation and is provided for comparison only. □

Example 16.3: *Door Detection Using Ultrasonic and Infrared Sensors.* Consider the problem of recognizing a door (an opening) wide enough for a mobile robot to move through. The mobile robot (TRC Labmate) is equipped with an array of four ultrasonic and four infrared Boolean sensors on each of four sides, as shown in Figure 16.1. We address only the problem of detecting a wide enough door

Table 16.2. Performance of Nadaraya–Watson estimator for decision fusion

Sample size	Test set	S_1	S_2	S_3	S_4	S_5	Nadaraya–Watson
100	100	7.0	20.0	33.0	35.0	55.0	12.0
1000	1000	11.3	18.5	29.8	38.7	51.6	10.6
10000	10000	9.5	20.1	30.3	39.8	49.6	8.58
50000	50000	10.0	20.1	29.8	39.9	50.1	8.860

© 2005 by Chapman & Hall/CRC

Table 16.3. Comparative performance

Sample size	Test size	Bayesian fuser	Empirical decision	Nearest neighbor	Nadaraya–Watson
100	100	91.91	23.00	82.83	88.00
1000	1000	91.99	82.58	90.39	89.40
10000	10000	91.11	90.15	90.81	91.42
50000	50000	91.19	90.99	91.13	91.14

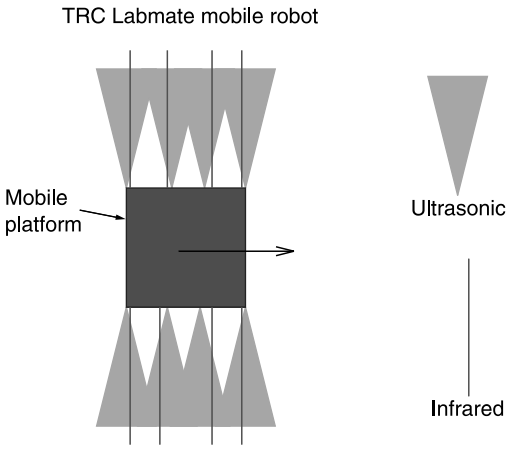


Figure 16.1. Schematic of sensory system (only the side sensor arrays are shown for simplicity).

when the sensor array of any side is facing it. The ultrasonic sensors return a measurement corresponding to distance to an object within a certain cone, as illustrated in Figure 16.1. The infrared sensors return Boolean values based on the light reflected by an object in the line-of-sight of the sensor; white, smooth objects are detected due to high reflectivity, while objects with black or rough surfaces are generally not detected. In practice, both ultrasonic and infrared sensors are unreliable, and it is very difficult to obtain accurate error distributions of these sensors. The ultrasonic sensors are susceptible to multiple reflections and the profiles of the edges of the door. The infrared sensors are susceptible to surface texture and color of the wall and edges of the door. Accurate derivation of probabilistic models for these sensors requires a detailed knowledge of the physics and engineering of the device as well as *a priori* statistical information. Consequently, a Bayesian solution to this problem is very hard to implement. On the other hand, it is relatively easy to collect experimental data by presenting to the robot doors that are wide enough as well as those that are narrower than the robot. We employ the Nadaraya–Watson estimator to derive a nonlinear relationship between the width of the door and the sensor readings. Here, the training sample is generated by actually recording the measurements while the sensor system is facing the door. Positive examples are generated if the door is wide enough for the robot and the sensory system is facing the door. Negative examples are generated when the door is not wide enough or the sensory system is not correctly facing a door (wide enough or not). The robot is manually located in various positions to generate the data. Consider the sensor array of a particular side of the mobile robot. Here, $Y^{(1)}, Y^{(2)}, Y^{(3)}, Y^{(4)}$ correspond to the normalized distance measurements from the four ultrasonic sensors, and $Y^{(5)}, Y^{(6)}, Y^{(7)}, Y^{(8)}$ correspond to the Boolean measurements of the infrared sensors. $X = 1$ if the sensor system is correctly facing a wide enough door, and is zero otherwise. The training data included 6 positive examples and 12 negative examples. The test data included three positive examples and seven negative examples. The Nadaraya–Watson estimator predicted the correct output in all examples of test data. □

© 2005 by Chapman & Hall/CRC

16.7 Performance of Fused System

We now address the issue of the relative performance of the composite system, composed of the fuser and $S_1, S_2, \ldots, S_N$, and the individual sensors or sensor subsets. We describe sufficiency conditions under which the composite system can be shown to be at least as good as the best sensor or best subset of sensors. In the empirical risk minimization methods $I_F(\hat{f})$ is shown to be close to $I_F(f^\star)$, which depends on $\mathcal{F}$. In general $I_F(f^\star)$ could be very large for particular fuser classes. Note that one cannot simply choose an arbitrary large $\mathcal{F}$; if so, the performance guarantees of the type in Equation (16.1) cannot be guaranteed. If $I_F(f^\star) > I(S_i)$, then fusion is not useful, since one is better off just using S_i. In practice, however, such a condition cannot be verified if the distributions are not known.

For simplicity, we consider a system of N sensors such that $X \in [0,1]$, $Y^{(i)} \in [0,1]$ and the *expected square error* is given by

$$I_S(S_i) = \int [X - Y^{(i)}]^2 \, dP_{Y^{(i)},X}$$

The *expected square error* of the fuser f is given by

$$I_F(f) = \int [X - f(Y)]^2 \, dP_{Y,X}$$

respectively, where $Y = \left(Y^{(1)}, Y^{(2)}, \ldots, Y^{(N)}\right)$.

16.7.1 Isolation Fusers

If the distributions are known, then one can derive the best sensor $S_{i^\star}$ such that $I_S(S_{i^\star}) = \min_{i=1}^N I_S(S_i)$. In the present formulation, the availability of *only* a sample makes the selection (with probability 1) of the best sensor infeasible, even in the special case of the target detection problem [15]. In this section, we present a method that circumvents this difficulty by fusing the sensors such that the performance of the best sensor is achieved as a minimum. The method is fully sample-based, in that no comparative performance of the sensors is needed — in particular, the best sensor may be unknown.

A function class $\mathcal{F} = \{f : [0,1]^k \mapsto [0,1]\}$ has the *isolation property* if it contains the functions $f^i(y_1, y_2, \ldots, y_k) = y_i$ for all $i = 1, 2, \ldots, k$. If $\mathcal{F}$ has the isolation property, we have

$$\begin{aligned} I_F(f^\star) &= \min_{f \in \mathcal{F}} \int (X - f(Y))^2 \, dP_{Y,X} \le \int \left(X - f^i(Y)\right)^2 dP_{Y,X} \\ &= \int \left(X - Y^{(i)}\right)^2 dP_{Y,X} = I_S(S_i) \end{aligned}$$

which implies $I_F(f^\star) = \min_{i=1}^N I_S(S_i) - \Delta$ for some $\Delta \in [0, \infty)$. Owing to the isolation property, we have $\Delta \ge 0$, which implies that the error of $f^\star$ is no higher than $I(S_{i^\star})$, but it can be significantly smaller. The precise value of Δ depends on $\mathcal{F}$, but the isolation property guarantees that $I_F(f^\star) \le \min_{i=1}^N I_S(S_i)$ as a minimum.

Let the set S be equipped with a pseudometric ρ. The *covering number* $N_C(\epsilon, \rho, S)$ under metric ρ is defined as the smallest number of closed balls of radius ϵ, and centers in S, whose union covers S. For a set of functions $\mathcal{G} = \{g : \Re^M \mapsto [0,1]\}$, we consider two metrics defined as follows: for $g_1, g_2 \in G$ we have

$$d_P(g_1, g_2) = \int_{z \in \Re^M} |g_1(z) - g_2(z)| \, dP$$

© 2005 by Chapman & Hall/CRC

for the probability distribution P defined on $\Re^M$, and

$$d_\infty(g_1, g_2) = \sup_{z \in \Re^M} |g_1(z) - g_2(z)|$$

This definition is applied to functions defined on $A \subseteq \Re^M$ by extending them to take value zero on $\Re^M \setminus A$.

Theorem 16.4 [40] *Consider a fuser class* $\mathcal{F} = \{f : [0,1]^N \mapsto [0,1]\}$, *such that* $I_F(f^\star) = \min_{f \in \mathcal{F}} I_F(f)$ *and* $\hat{I}_F(\hat{f}) = \min_{f \in \mathcal{F}} \hat{I}_F(f)$. *If* $\mathcal{F}$ *has the isolation property, we have*

$$I_F(f^\star) = \min_{i=1}^{N} I_S(S_i) - \Delta$$

for $\Delta \in [0, \infty)$, *and*

$$P^l_{Y,X}\left[I_F(\hat{f}) - \min_{i=1}^{N} I_S(S_i) + \Delta > \epsilon \right] < \delta$$

given the sample size l *of at least*

$$\frac{2048}{\epsilon^2}[\ln N_C(\epsilon/64, \mathcal{F}) + \ln(4/\delta)]$$

for cases: (i) $N_C(\epsilon, \mathcal{F}) = N_C(\epsilon, d_\infty, \mathcal{F})$, *and (ii)* $N_C(\epsilon, \mathcal{F}) = N_C(\epsilon, d_P, \mathcal{F})$ *for all distributions* P.

If $\mathcal{F}$ has the isolation property, then the fuser is guaranteed to perform at least as good as the best sensor in the PAC sense. No information other than the iid sample is needed to ensure this result. Since $\Delta \geq 0$, under the sample size of Theorem 16.4, we trivially have

$$P\left[I_F(\hat{f}) - \min_{i=1}^{N} I_S(S_i) > \epsilon \right] < \delta$$

The sample size needed is expressed in terms of d_∞ or distribution-free covers for $\mathcal{F}$. For smooth fusers, such as sigmoid neural networks, we have simple d_∞ cover bounds. In other cases, the pseudo-dimension and scale-sensitive dimension of $\mathcal{F}$ provide the distribution-free cover bounds needed in Theorem 16.4. The isolation property was first proposed in [23,41] for concept and sensor fusion problems. For linear combinations, i.e. $f(y_1, y_2, \ldots, y_k) = w_1 y_1 + w_2 y_2 + \cdots + w_k y_k$, for $w_i \in \Re$, this property is trivially satisfied. For potential functions [28] and feedforward sigmoid networks [42], this property is not satisfied in general; see [43] for a more detailed discussion on the isolation property and various function classes that have this property.

Consider the special case where S_i values are classifiers obtained using different methods, as in [44]. For Boolean functions, the isolation property is satisfied if $\mathcal{F}$ contains all Boolean functions on k variables. S_i computed based an iid l-sample is *consistent* if $I_S(S_i) \to I_S(S^\star)$, where $S^\star$ is the Bayes classifier. By the isolation property, if one of the classifiers is consistent, the fused classifier system (trained by l-sample independent from n-sample used by the classifiers) can be seen to be consistent. Such a result was obtained in [44] for linear combinations (for which $N_C(\epsilon, \mathcal{F})$ is finite). The above result does not require the linearity, but pinpoints the essential property, namely the isolation.

© 2005 by Chapman & Hall/CRC

Linear fusers have been extensively used as fusers in combining neural network estimators [45], regression estimators [11,46], and classifiers [44]. Since the linear combinations possess the isolation property, Theorem 16.4 provides some analytical justification for these methods.

16.7.2 Projective Fusers

A *projective fuser* [44], f_P, corresponding to a *partition* $P = \{\pi_1, \pi_2, \ldots, \pi_k\}$, $k \leq N$, of input space $\Re^d$ ($\pi_i \subseteq [0,1]^d$, $\bigcup_{i=1}^{k} \pi_i = \Re^d$, and $\pi_i \cap \pi_j = \phi$ for $i \neq j$), assigns each block π_i to a sensor S_j such that

$$f_P(Y) = Y^{(j)}$$

for all $X \in \pi_i$, i.e. the fuser simply transfers the output of the sensor S_j for every point in π_i. An *optimal projective fuser*, denoted by f_{P^*}, minimizes $I(.)$ over all projective fusers corresponding to all partitions of $\Re^d$ and assignments of blocks to sensors $S_1, S_2, \ldots, S_N$.

We define the *error regression* of the sensor S_i and fuser f_F as

$$\mathcal{E}(X, S_i) = \int C(X, Y^{(i)})\, \mathrm{d}P_{Y|X}$$

and

$$\mathcal{E}(X, f_P) = \int C(X, f_P(Y))\, \mathrm{d}P_{Y|X},$$

respectively. The *projective fuser* based on the lower envelope of error regressions of sensors is defined by

$$f_{LE}(Y) = Y^{(i_{LE}(X))}$$

where

$$i_{LE}(X) = \arg \min_{i=1,2,\ldots,N} \mathcal{E}(X, S_i)$$

We have $\mathcal{E}(X, f_{LE}) = \min_{i=1,\ldots,N} \mathcal{E}(X, S_i)$, or equivalently the error regression of f_{LE} is the lower envelope with respect to X of the set of error regressions of sensors given by $\{\mathcal{E}(X, S_1), \ldots, \mathcal{E}(X, S_N)\}$.

Example 16.4: [47] Consider that X is uniformly distributed over $[0, 1]$, which is measured by two sensors S_1 and S_2. Let $C(X, Y^{(i)}) = (X - Y^{(i)})^2$. Consider $Y^{(1)} = X + |X - 1/2| + U$ and $Y^{(2)} = X + 1/[4(1 + |X - 1/2|)] + U$, where U is an independent random variable with zero mean. Thus, for both sensors the measurement error at any X is represented by U. Note that

$$E[Y^{(1)} - X] = |X - 1/2|$$

$$E[Y^{(2)} - X] = 1/[4(1 + |X - 1/2|)]$$

Thus, S_1 achieves a low error in the middle of the range $[0, 1]$, and S_2 achieves a low error toward the end point of the range $[0, 1]$. The error regressions of the sensors are given by

$$\mathcal{E}(X, S_1) = (X - 1/2)^2 + E[U^2]$$

$$\mathcal{E}(X, S_2) = 1/[16(1 + |X - 1/2|)^2] + E[U^2]$$

© 2005 by Chapman & Hall/CRC

We have

$$I(S_1) = 0.0833 + E[U^2] \quad \text{and} \quad I(S_2) = 0.125 + E[U^2]$$

which indicates that S_1 is the better of the two sensors. Now consider the projective fuser f_{LE} specified as follows, which corresponds to the lower envelope of $\mathcal{E}(X, S_1)$ and $\mathcal{E}(X, S_2)$:

Range for X	Sensor to be projected
[0, 0.134]	S_2
[0.134, 0.866]	S_1
[0.866, 1]	S_2

Then, we have $I(f_{\text{LE}}) = 0.0828 + E[U^2]$, which is lower than that of the best sensor. □

Example 16.5: [47] We consider a classification example such that $X \in [0, 1] \times \{0, 1\}$ is specified by a function $f_X = 1_{[1/4, 3/4]}$, where $1_A(z)$ is the *indicator function* (which has a value 1 if and only if $z \in A$ and has value 0 otherwise). The value of X is generated as follows: a random variable Z is generated uniformly in the interval $[0, 1]$ as the first component, and then $f_X(Z)$ forms the second component, i.e. $X = (Z, f_X(Z))$. In the context of the detection problem, the second component of X corresponds to the presence ($f_X(Z) = 1$) or absence ($f_X(Z) = 0$) of a target, which is represented by a feature Z taking a value in the interval $[1/4, 3/4]$. Each sensor consists of a device to measure the first component of X and an algorithm to compute the second component. We consider that S_1 and S_2 have ideal devices that measure Z without an error, but make errors in utilizing the measured features. Consider that $Y^{(1)} = (Z, 1_{[1/4-\epsilon_1, 3/4]}(Z))$ and $Y^{(2)} = (Z, 1_{[1/4, 3/4-\epsilon_2]}(Z))$ for some $0 < \epsilon_1, \epsilon_2 < 1/4$ (see Figure 16.2). In other words, there is no measurement noise in the sensors but just a systematic error due to how the feature value is utilized; addition of independent measurement noise, as in Example 16.1, does not change the basic conclusions of the example. Now consider the quadratic cost function $C(X, Y^{(i)}) = (X - Y^{(i)})^{\text{T}}(X - Y^{(i)})$. The error regressions are given by $\mathcal{E}(X, S_1) = 1_{[1/4-\epsilon_1, 1/4]}(Z)$ and $\mathcal{E}(X, S_2) = 1_{[3/4-\epsilon_2, 3/4]}(Z)$, which corresponds to disjoint intervals of Z as shown in Figure 16.3. The lower envelope of the two regressions is the zero function, hence $I(f_{\text{LE}}) = 0$, whereas both $I(S_1)$ and $I(S_2)$ are positive. The profile of f_{LE} is shown at the bottom of Figure 16.2, wherein S_1 and S_2 are projected based on the first component of X in the intervals $[3/4 - \epsilon_2, 3/4]$ and $[1/4 - \epsilon_1, 1/4]$ respectively, and in other regions either sensor can be projected. □

The projective fuser based on error regressions is optimal as in the following theorem.

Theorem 16.5: [47] *The projective fuser based on the lower envelope of error regressions is optimal among all projective fusers.*

A special case of this theorem for function estimation can be found in [48], and one for classifiers can be found in [49]. A sample-based version of the projective fuser is presented for function estimation using the nearest neighbor concept in [50], where finite-sample performance bounds are derived using the total variances of various individual estimators. Furthermore, f_{LE} may not be optimal in a larger class of fusers where some function of the sensor output (as opposed to just the output) can be projected [47].

Example 16.3: [47] In Example 16.5, consider $f_X = 1_{[1/4, 3/4]}$,

$$Y^{(1)}(X) = (Z, 1_{[1/4-\epsilon_1, 3/4-\epsilon_1]}(Z))$$

$$Y^{(2)}(X) = (Z, 1_{[1/4, 3/4-\epsilon_2]}(Z))$$

© 2005 by Chapman & Hall/CRC

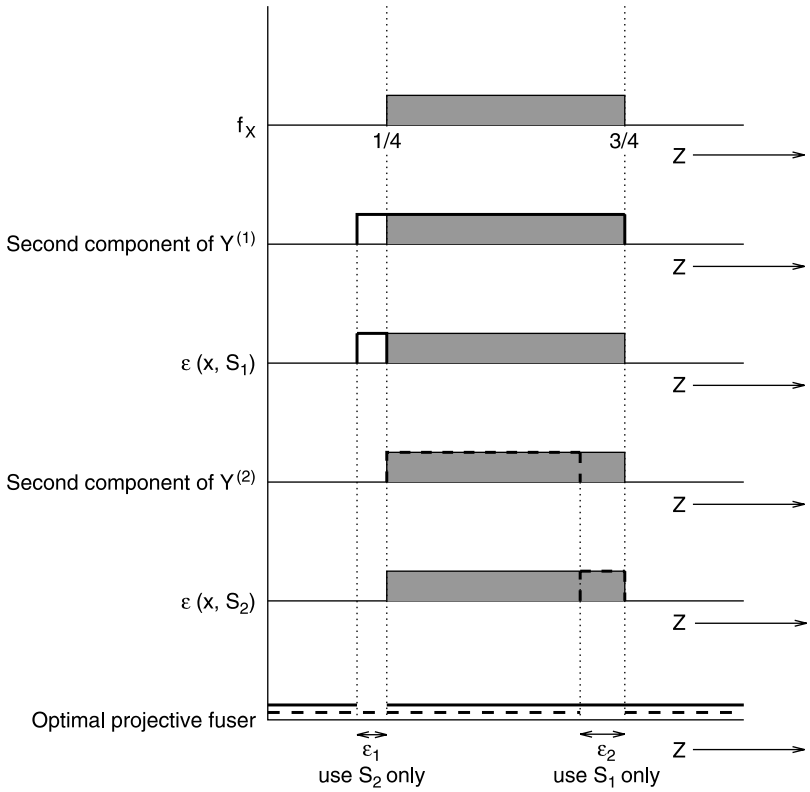


Figure 16.2. Illustration for example.

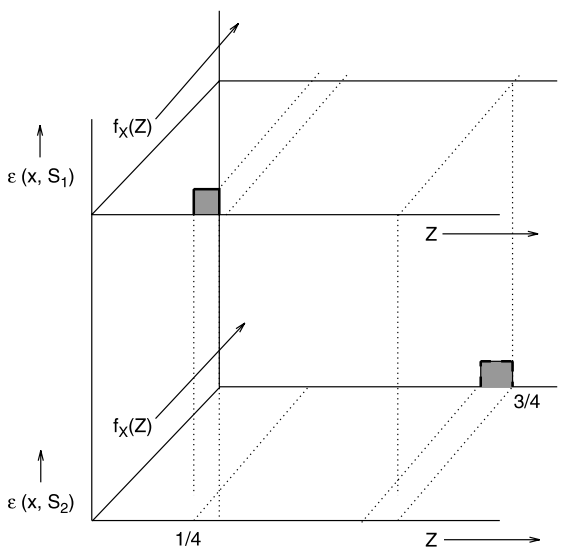


Figure 16.3. Illustration of error regressions.

© 2005 by Chapman & Hall/CRC

for some $0 < \epsilon_1, \epsilon_2 < 1/8$, and $\epsilon_1 < \epsilon_2$. Thus, we have $\mathcal{E}(X, S_1) = 1_{[1/4-\epsilon_1, 1/4]}(Z)$ and $\mathcal{E}(X, S_2) = 1_{[3/4-\epsilon_2, 3/4]}(Z)$, whose lower envelope is not the zero function. Thus, we have $\mathcal{E}(X, f_{\mathrm{LE}}) = 1_{[3/4-\epsilon_2, 3/4-\epsilon_1]}(Z)$ and $I(f_{\mathrm{LE}}) = \int_{[3/4-\epsilon_2, 3/4-\epsilon_1]} \mathrm{d}P_Z$. By changing the assignment $Y^{(1)}$ of f_{LE} to $1 - Y^{(1)}$ for $Z \in [3/4 - \epsilon_2, 3/4 - \epsilon_1]$, one can easily achieve zero error. □

16.8 Metafusers

In this section, we first show that the projective and linear fusers offer complementary performances, which leads us to the idea of combining them to exploit their relative merits. Such an approach leads to the concept of *metafusers*, and here we describe how the isolation property can be utilized to design them.

The output of a linear fuser corresponding to input X and sensor output $Y = (Y^{(1)}, \ldots, Y^{(N)})$ is defined as

$$f_{\mathrm{L}}(Y) = \sum_{i=1}^{N} \alpha_i Y^{(i)}$$

where α_i is a $d \times d$ matrix. For simplicity, we consider the case $d = 1$ such that $(\alpha_1, \ldots, \alpha_N) \in \Re^N$. An *optimal linear combination fuser*, denoted by f_{L^*}, minimizes $I(.)$ over all linear combinations. In general, f_{LE} is better than f_{L^*} if the individual sensors perform better in certain localized regions of $\Re^d$. On the flip side, if the sensors are equally distributed around certain values in a global sense, then f_{L} performs better, as illustrated below.

In Example 16.5, for $f_{\mathrm{L}} = \alpha_1 Y^{(1)} + \alpha_2 Y^{(N)}$, we have

$$I(f_{\mathrm{L}}) = \alpha_1^2 \int_{[1/4-\epsilon_1, 1/4)} \mathrm{d}P_Z + (1 - \alpha_1 - \alpha_2)^2 \int_{[1/4, 3/4-\epsilon_2)} \mathrm{d}P_Z + (1 - \alpha_1)^2 \int_{[3/4-\epsilon_2, 3/4]} \mathrm{d}P_Z$$

which is nonzero no matter what the coefficients are. The error regressions of S_1 and S_2 take nonzero values in the intervals $[1/4 - \epsilon_1, 1/4]$ and $[3/4 - \epsilon_2, 3/4]$ of Z, respectively. Since, these intervals are disjoint, there is no possibility of the error of one sensor being canceled by a scaler multiplier of the other. This argument is true in general: if the error regressions of the sensors take nonzero values on disjoint intervals, then any linear fuser will have nonzero residual error. On the other hand, the disjointness yields $\mathcal{E}(X, f_{\mathrm{LE}}) = 0$, for all X, and hence $I(f_{\mathrm{LE}}) = 0$. Consider that in Example 16.5, $f_X = 1$ for $Z \in [0, 1]$,

$$Y^{(1)}(X) = (Z, \epsilon Z + 1 - \epsilon)$$

$$Y^{(2)}(X) = (Z, -\epsilon Z + 1 + \epsilon)$$

for $0 < \epsilon < 1$. The optimal linear fuser is given by $f_{\mathrm{L}^*}(Y) = 1/2(Y^{(1)} + Y^{(2)}) = 1$, and $I(f_{\mathrm{L}^*}) = 0$. At every $X \in [0, 1]$, we have

$$\mathcal{E}(X, S_1) = \mathcal{E}(X, S_2) = \epsilon^2 (1 - Z)^2 = \mathcal{E}(X, f_{\mathrm{LE}})$$

Thus, $I(f_{\mathrm{LE}}) = \epsilon^2 \int_{[0, 1]} (1 - Z)^2 \, \mathrm{d}P_Z > 0$, whereas $I(f_{\mathrm{L}^*}) = 0$.

Thus, the performances of the optimal linear and projective fusers are complementary in general. We now combine linear and projective fusers to realize various metafusers that are guaranteed to be at least as good as the best sensor as well as best sensor. By including the optimal linear combination as S_{N+1}, we can guarantee that $I(f_{\mathrm{LE}}) \leq I(f_{\mathrm{L}^*})$ by the isolation property of projective fusers [47]. Since linear

© 2005 by Chapman & Hall/CRC

combinations also satisfy the isolation property, we in turn have $I(f_{L^*}) \leq \min_{i=1,\ldots,N} I(S_i)$. The roles of f_{L^*} and f_{LE} can be switched — by including f_{LE} as one of the components of f_{L^*} — to show that

$$I(f_{L^*}) \leq I(f_{\mathrm{LE}}) \leq \min_{i=1,\ldots,N} I(S_i)$$

One can design a metafuser by utilizing the available sensors which are combined using a number of fusers including a fuser based on the isolation property (e.g. a linear combination). Consider that we employ a metafuser based on a linear combination of the fusers. Then, the fused system is guaranteed to be at least as good as the best of the fusers as well as the best sensor. If, at a later point, a new sensor or a fuser is developed, then it can be easily integrated into the system by retraining the fuser and/or metafuser as needed. As a result, we have a system guaranteed (in PAC sense) to perform at least as good as the best available sensor and fuser at all times. Also, the computational problem of updating the fuser and/or metafuser is a simple least-squares estimation that can be solved using a number of available methods.

16.9 Conclusions

In a DSN, we considered that the sensor outputs are related to the actual feature values according to a probability distribution. For such a system, we presented an overview of informational and computational aspects of a fuser that combines the sensor outputs to more accurately predict the feature when the sensor distributions are unknown but iid measurements are given. Our performance criterion is the probabilistic guarantee in terms of distribution-free sample bounds based entirely on a finite sample. We first described two methods, based on the empirical risk minimization approach, which yield a fuser that is guaranteed, with a high probability, to be close to an optimal fuser. Note that the optimal fuser is computable only under a complete knowledge of sensor distributions. Then we described the isolation fusers that are guaranteed to perform at least as good as the best sensor. We then described the projective fusers that are guaranteed to perform at least as good as the best subset of sensors. We briefly discussed the notion of metafusers that can combine fusers of different types.

The overall focus of this chapter is very limited: we only considered sample-based fuser methods that provide finite sample performance guarantees. Even then, there are a number of important issues for the fuser rule computation that have not been covered here. An important aspect is the utilization of fusers that have been designed for known distribution cases for the sample-based case. In many important cases, the fuser formulas expressed in terms of probabilities can be converted into sample-based ones by utilizing suitable estimators [13]. It would be interesting to see an application of this approach to the generic sensor fusion problem. For the most part, we only considered stationary systems, and it would of future interest to study sample-based fusers for time-varying systems.

Acknowledgment

This research is sponsored by the Material Science and Engineering Division, Office of Basic Energy Sciences, U.S. Department of Energy, under Contract No. DE-AC05-00OR22725 with UT-Battelle, LLC, the Defense Advanced Projects Research Agency under MIPR No. K153, and by National Science Foundation under Grants No. ANI-0229969 and No. ANI-335185.

References

[1] Varshney, P.K., *Distributed Detection and Data Fusion*, Springer-Verlag, 1997.
[2] Vapnik, V., *Estimation of Dependences Based on Empirical Data*, Springer-Verlag, New York, 1982.
[3] Grofman, B. and Owen, G. (eds.), *Information Pooling and Group Decision Making*, Jai Press Inc., Greenwich, CT, 1986.

© 2005 by Chapman & Hall/CRC

[4] De Laplace, P.S., Deuxiéme supplèment á la thèorie analytique des probabilitès, 1818. Reprinted (1847) in *Oeuvres Complétes de Laplace*, vol. 7, Gauthier-Villars, Paris, 531.
[5] Madan, R.N. and Rao, N.S.V., Guest editorial on information/decision fusion with engineering applications, *Journal of Franklin Institute*, 336B(2), 199, 1999.
[6] Von Neumann, J., Probabilistic logics and the synthesis of reliable organisms from unreliable components, in Shannon, C.E. and McCarthy, J. (eds), *Automata Studies*, Princeton University Press, 1956, 43.
[7] Granger, C.W.J., Combining forecasts — twenty years later, *Journal of Forecasting*, 8, 167, 1989.
[8] Chow, C.K., Statistical independence and threshold functions, *IEEE Transactions on Electronic Computers*, EC-16, 66, 1965.
[9] Hashem, S. *et al.*, Optimal linear combinations of neural networks: an overview, in *Proceedings of 1994 IEEE Conference on Neural Networks*, 1507, 1994.
[10] Dasarathy, B.V., *Decision Fusion*, IEEE Computer Society Press, Los Alamitos, CA, 1994.
[11] Breiman, L., Stacked regressions, *Machine Learning*, 24(1), 49, 1996.
[12] Drakopoulos, E. and Lee, C.C., Optimal multisensor fusion of correlated local decision, *IEEE Transactions on Aerospace Electronics Systems*, 27(4), 593, 1991.
[13] Rao, N.S.V., Distributed decision fusion using empirical estimation, *IEEE Transactions on Aerospace and Electronic Systems*, 33(4), 1106, 1996.
[14] Rao, N.S.V., On sample-based implementation of non-smooth decision fusion functions, in *IEEE/SICE/RSJ International Conference on Multisensor Fusion and Integration for Intelligent Systems*, 2001.
[15] Devroye, L. *et al.*, *A Probabilistic Theory of Pattern Recognition*. Springer-Verlag, New York, 1996.
[16] Rao, N.S.V. and Oblow, E.M., Majority and location-based fusers for PAC concept learners. *IEEE Transactions on Systems, Man and Cybernetics*, 24(5), 713, 1994.
[17] Rao, N.S.V. and Oblow, E.M., *N*-learners problem: System of PAC learners, in *Computational Learning Theory and Natural Learning Systems, Vol IV: Making Learning Practical*, MIT Press, 1997, 189.
[18] Anthony, M., Probabilistic analysis of learning in artificial neural networks: the PAC model and its variants. NeuroCOLT Technical Report Series NC-TR-94-3, Royal Holloway, University of London, 1994.
[19] Tang, Z. and Koehler, G.J., Lipschitz properties of feedforward neural networks, Technical report, 1994.
[20] Rao, N.S.V., Fusion rule estimation in multiple sensor systems using training, in *Modelling and Planning for Sensor Based Intelligent Robot Systems*, Bunke, H. *et al.* (eds), World Scientific, 1995, 179.
[21] Rao, N.S.V., Multiple sensor fusion under unknown distributions. *Journal of Franklin Institute*, 336(2), 285, 1999.
[22] Rao, N.S.V., Fusion methods in multiple sensor systems using feedforward neural networks. *Intelligent Automation and Soft Computing*, 5(1), 21, 1999.
[23] Rao, N.S.V., Fusion methods for multiple sensor systems with unknown error densities, *Journal of Franklin Institute*, 331B(5), 509, 1994.
[24] Lugosi, G. and Zeger, K., Nonparametric estimation via empirical risk minimization, *IEEE Transactions on Information Theory*, 41(3), 677, 1995.
[25] Sima, J., Back-propagation is not efficient, *Neural Networks*, 9(6), 1017, 1996.
[26] Rao, N.S.V., Vector space methods for sensor fusion problems, *Optical Engineering*, 37(2), 499, 1998.
[27] Vavasis, S.A., *Nonlinear Optimization*, Oxford University Press, New York, 1991.
[28] Aizerman, M.A. *et al.*, Extrapolative problems in automatic control and method of potential functions, *American Mathematical Society Translations*, 87, 281, 1970.
[29] Rao, N.S.V. *et al.*, Iyengar. Learning algorithms for feedforward networks based on finite samples. *IEEE Transactions on Neural Networks*, 7(4), 926, 1996.

© 2005 by Chapman & Hall/CRC

[30] Kurkova, V., Kolmogorov's theorem and multilayer neural networks, *Neural Networks*, 5, 501, 1992.
[31] Cybenko, G., Approximation by superpositions of a sigmoidal function, *Mathematics of Contols, Signals, and Systems*, 2, 303, 1989.
[32] Ciesielski, Z., Haar system and nonparametric density estimation in several variables, *Probability and Mathematical Statistics*, 9, 1, 1988.
[33] Prakasa Rao, B.L.S., *Nonparametric Functional Estimation*, Academic Press, New York, 1983.
[34] Engel, J., A simple wavelet approach to nonparametric regression from recursive partitioning schemes, *Journal of Multivariate Analysis*, 49, 242, 1994.
[35] Nadaraya, E.A., *Nonparametric Estimation of Probability Densities and Regression Curves*, Kluwer Academic Publishers, Dordrecht, 1989.
[36] Rao, N.S.V. and Protopopescu, V., On PAC learning of functions with smoothness properties using feedforward sigmoidal networks. *Proceedings of the IEEE*, 84(10), 1562, 1996.
[37] Rao, N.S.V., Nadaraya–Watson estimator for sensor fusion, *Optical Engineering*, 36(3), 642, 1997.
[38] Preparata, F.P. and Shamos, I.A., *Computational Geometry: An Introduction*, Springer-Verlag, New York, 1985.
[39] Rao, N.S.V. and Iyengar, S.S., Distributed decision fusion under unknown distributions, *Optical Engineering*, 35(3), 617, 1996.
[40] Rao, N.S.V., A fusion method that performs better than best sensor, in *First International Conference on Multisource–Multisensor Data Fusion*, 1998, 19.
[41] Rao, N.S.V. *et al.*, *N*-learners problem: fusion of concepts. *IEEE Transactions on Systems, Man and Cybernetics*, 24(2), 319, 1994.
[42] Roychowdhury, V. *et al.* (eds), *Theoretical Advances in Neural Computation and Learning*, Kluwer Academic, Boston, 1994.
[43] Rao, N.S.V., Finite sample performance guarantees of fusers for function estimators, *Information Fusion*, 1(1), 35, 2000.
[44] Mojirsheibani, M., A consistent combined classification rule. *Statistics and Probability Letters*, 36, 43, 1997.
[45] Hashem, S., Optimal linear combinations of neural networks, *Neural Networks*, 10(4), 599, 1997.
[46] Taniguchi, M. and Tresp, V., Averaging regularized estimators, *Neural Computation*, 9, 1163, 1997.
[47] Rao, N.S.V., Projective method for generic sensor fusion problem, in *IEEE/SICE/RSJ International Conference on Multisensor Fusion and Integration for Intelligent Systems*, 1999, 1.
[48] Rao, N.S.V., On optimal projective fusers for function estimators, in *Second International Conference on Information Fusion*, 1999, 296.
[49] Rao, N.S.V., To fuse or not to fuse: fuser versus best classifier, in *SPIE Conference on Sensor Fusion: Architectures, Algorithms, and Applications II*, 34, 1998, 25.
[50] Rao, N.S.V., Nearest neighbor projective fuser for function estimation, in *Proceedings of International Conference on Information Fusion*, 2002.

© 2005 by Chapman & Hall/CRC

17

Soft Computing Techniques

R.R. Brooks

17.1 Problem Statement

When trade-offs in an implementation have been identified, many design issues become optimization problems. A very large class of combinatorial optimization problems can be shown to be NP-hard, so that large instances of the problems cannot be solved given current technologies. There is a good chance that it may never be tractable to find the optimal solutions to any of these problems within a reasonable period of time. In spite of this, some techniques have been shown capable of tractably finding near-optimal or reasonable answers to these problems. These techniques are usually referred to as heuristics. The general classes of techniques given here are thus known as meta-heuristics. Meta-heuristics, fuzzy sets, and artificial neural networks also make up the set of what has come to be known as soft computing technologies. These are advanced computational techniques for problem solving.

This chapter presents a brief introduction to genetic algorithms, linear programming, simulated annealing, tabu search, TRUST, artificial neural networks, and fuzzy sets. An example that uses some of these techniques can be found in the chapter on data registration (Chapter 20). More in-depth treatments of this work can be found in [1–4].

17.2 Genetic Algorithms

Genetic algorithms, first developed in the mid-1960s, apply a Darwinian concept of survival of the fittest to optimization problems. Refer to Holland [5] for details. An in-depth comparison of genetic algorithms versus exhaustive search for a reliability design problem can be found in Kumar *et al.* [6]. Several different reproduction strategies are attempted in the literature. The metaphorical approach of genetic search has been shown experimentally to be useful for solving many difficult optimization problems.

Possible solutions to a problem are called chromosomes, and a diverse set of chromosomes is grouped into a gene pool. The relative quality of these answers is determined using a fitness function, and this quality is used to determine whether or not the chromosomes will be used in producing the next generation of chromosomes. The contents of high-quality solutions are more likely to continue into the next generation. The next generation is generally formed via the processes of crossover, i.e. combining elements of two chromosomes from the gene pool, and mutation, i.e. randomly altering elements of a chromosome.

© 2005 by Chapman & Hall/CRC

A large number of strategies exist for determining the contents of a new generation. Two different genetic algorithms are discussed here. They differ only in their reproduction strategies. The first strategy has been described by Holland [5]. Each string in the gene pool is evaluated by the fitness function. Based on the quality of the answer represented by the string, it is assigned a probability of being chosen for the pool of strings used to produce the next generation. Those with better answers are more likely to be chosen. A mating pool is then constructed by choosing strings at random from the gene pool following the probability distribution derived.

The new generation is formed by mixing the elements of two strings in the mating pool chosen at random. This is generally called crossover. Different crossover probabilities may be used. Where the string is split can be chosen at random, or deterministic schemes are possible as well. A certain amount of mutation usually exists in the system, where one or more elements at random are replaced by random values.

A second strategy that may be applied has been described by Bean [7]. This strategy is described as elitist, since some percentage of the strings with the best fitness function values are copied directly into the next generation. In addition to this, in our work [1–3], some of the strings for the next generation are the result of random mutations. Random mutations may be strings where all elements are chosen at random. Performing crossover between random strings in the current generation forms the rest of the new generation. The choice is done entirely at random; no weighting based on the quality of the string is performed. Bean [7] reports that this strategy has been found to be stable experimentally. Its implementation is straightforward.

Genetic algorithms are not sensitive to the presence of local minima since they work on a large number of points in the problem space simultaneously. By comparing many possible solutions they achieve what Holland [5] has termed implicit parallelism, which increases the speed of their search for an optimal solution. Discussion of the advantages gained by using genetic algorithms instead of exhaustive search for a different optimization problem based on system reliability can be found in Painton and Campbell [8].

17.3 Simulated Annealing

Simulated annealing attempts to find optimal answers to a problem in a manner analogous to the formation of crystals in cooling solids. A material heated beyond a certain point will become fluid; if the fluid is cooled slowly then the material will form crystals and revert to a minimal energy state. Refer to Laarhoven and Aarts [9] for a full description of simulated annealing and a discussion of its scientific basis.

The strategy of the algorithm is again based on a *fitness function* comparing the relative merits of various points in the problem space. As before, vectors corresponding to possible system configurations describe the problem space. The fitness function is the value of the configuration described by the vector. The algorithm starts at a point in the search space. From the algorithm's current position, a neighboring point is chosen at random. The cost difference between the new point and the current point is calculated. This difference is used together with the current system temperature to calculate the probability of the new position being accepted. This probability is given by a Boltzmann distribution $e^{-\Delta C/\tau}$. The process continues with the same temperature τ for either a given number of iterations, or until a given number of positions have been occupied, at which time the value τ is decreased. The temperature decreases until no transitions are possible, so the system remains frozen in one position. This occurs only when $\triangle C$ is positive for all neighboring points; therefore, the position must be a local minimum and may be the global minimum [10].

The simulated annealing method used in our research is based on the algorithm given in Laarhoven and Aarts [9]. The algorithm was always modified so that the parameters being optimized and the fitness function are appropriate for our application, and a cooling schedule was found that allows the algorithm to converge to a reasonable solution.

© 2005 by Chapman & Hall/CRC

Just as many different reproduction schemes exist for genetic algorithms, several possible *cooling schedules* exist for simulated annealing. A cooling schedule is defined by the initial temperature, the number of iterations performed at each temperature, the number of position modifications allowed at a given temperature, and the rate of decrease of the temperature. The answers found by the algorithm are directly dependent on the cooling schedule and no definite rules exist for defining the schedule [9]. The cooling schedule is important, in that it determines the rate of convergence of the algorithm as well as the quality of the results obtained. The complexity of this approach could potentially increase in the order of J^2. This increase is significantly less than the exponential growth of the exhaustive search, but is greater than the increase for the genetic search.

17.4 Trust

The global minimization problem can be stated as follows: given a function f over some (possibly vector-valued) domain D, compute $\overline{x}_{GM} \in D$ such that $f(\overline{x}_{GM}) \leq f(\overline{x})$, $\forall \overline{x} \in D$. Usually, but not necessarily, f is assumed to be continuous and differentiable.

One strategy for global minimum determination is shown in Figure 17.1. Given an initial starting position in the search space, find the local minimum closest to this value. This can be done using gradient descent or a probabilistic search [11], such as genetic algorithms and simulated annealing. Once a local minimum is found, it is reasonable to assume that it is not globally optimal. Therefore, attempt to escape the local basin of values.

Global optimization research concentrates on finding methods for escaping the basin of attraction of a local minimum. Often, the halting condition is difficult to determine. This means there is a tradeoff between accepting a local minimum as the solution and performing an exhaustive search of the state space. In certain cases, global optimization can become prohibitively expensive when the search space has a large number of dimensions. Thus, there is a natural tradeoff between solution accuracy and the time spent searching. Additionally, if an analytic form of the cost function is unknown, as is typically the case, then this problem is more pronounced.

The following characteristics are desirable in a global optimization method:

1. Avoid entrapment in local minimum basins.
2. Avoid performing an exhaustive search on the state space.
3. Minimize the number of object function evaluations.
4. Have a clearly defined stopping criterion.

In practice, a good global optimization method judiciously balances these conflicting goals.

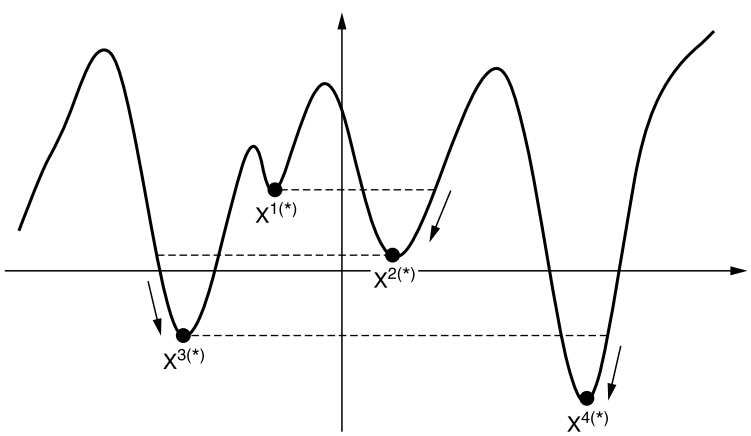


Figure 17.1. Tunneling method for determining global minimum.

© 2005 by Chapman & Hall/CRC

A number of approaches have been proposed for solving global optimization problems (see references in [12]). TRUST [12,13] is a deterministic global optimization method that avoids local minima entrapment and exhaustive search. This method defines a dynamic system using two concepts to avoid being stuck in local minima:

1. Subenergy tunneling
2. Non-Lipschitz terminal repellers

TRUST makes the following assumptions about the cost function f and its domain D [14]:

1. f: $D \rightarrow R$ is a lower semicontinuous function with a finite number of discontinuities.
2. $D \subseteq R^n$ is compact and connected.
3. Every local minimum of f in D is twice differentiable. Furthermore, for any local minima x_{LM} we have

$$\left.\frac{\partial f}{\partial x_i}\right|_{x_{\mathrm{LM}}} = 0, \forall i = 1, 2, \ldots, d \quad \text{and} \quad y^{\mathrm{T}} \frac{\partial^2 f(x_{\mathrm{LM}})}{\partial x^2} y \geq 0, \forall y \in D \tag{17.1}$$

where $\partial^2 f(x_{\mathrm{LM}})/\partial x^2$ is the Jacobian matrix given by

$$\left[\left.\frac{\partial^2 f}{\partial x_i \partial x_j}\right|_{x_{\mathrm{LM}}}\right]$$

TRUST uses tunneling [15] to avoid being trapped in local minima. Tunneling is performed by transforming the function f into a new function $E(x, x^*)$ with similar extrema properties, such that the current local minimum of f at some value x^* is a global maximum of $E(x, x^*)$. A value of f strictly less than $f(x^*)$ is then found by applying gradient descent. The general algorithm is as follows:

1. Use a gradient-descent method to find a local minimum at x^*.
2. Transform f into the following *virtual objective function*

$$E(x, x^*) = E_{\mathrm{sub}}(x, x^*) + E_{\mathrm{rep}}(x, x^*) \tag{17.2}$$

where

$$E_{\mathrm{sub}}(x, x^*) = \log\left[\frac{1}{1 + \mathrm{e}^{-(f(x)-f(x^*))+\mathrm{a}}}\right] \tag{17.3}$$

and

$$E_{\mathrm{rep}}(x, x^*) = -\frac{3}{4}\rho(x - x^*)^{4/3} u(f(x) - f(x^*)) \tag{17.4}$$

E_{sub} is the *subenergy tunneling term* and is used to isolate the function range of f less than the functional value $f(x^*)$. The difference $f(x) - f(x^*)$ offsets the f such that $f(x^*)$ tangentially intersects the x-axis. E_{rep} is the *terminal repeller term* and is used to guide the gradient-descent search in the next step. In this term, $u(y)$ is the Heaviside function. Note that $E(x, x^*)$ is a well-defined function with a global maximum at x^*.

© 2005 by Chapman & Hall/CRC

3. Apply gradient descent to $E(x, x^*)$. This yields the dynamical system

$$\dot{x} - \frac{\partial E}{\partial x} = -\frac{\partial f}{\partial x}\frac{1}{1 + e^{(f(x)-f(x^*))+a}} + \rho(x - x^*)^{1/3}u(f(x) - f(x^*)) \tag{17.5}$$

   An equilibrium state of $\dot{x}$ is a local minimum of $E(x, x^*)$ which in turn is a local or global minimum of the original function f.
4. Until the search boundaries are reached, repeat step 2 with the new local minima found in step 3 as the initial extrema.

Complete theoretical development of E_{sub} and E_{rep}, including a discussion of terminal repeller dynamics, can be found in Barhen and Protopopescu [14].

In the one-dimensional case, TRUST guarantees convergence to the globally optimal value. The reason for this is the transformation of f into $E(x, x^*)$ and the subsequent gradient-descent search generates monotonically decreasing minimal values of f (see Figures 17.2 and 17.3). This behavior is a result of the compactness of the domain D, since the function is not allowed to diverge to infinity. When the last local minimum is found and the method attempts to search $E(x, x^*)$, the difference $f(x) - f(x^*)$ becomes equivalent to the x-axis. Thus, $E(x, x^*) = E_{rep}(x, x^*)$, and the subsequent search on this curve will proceed until the endpoints of the domain D are reached.

In the more difficult multi-dimensional case, there is no theoretical guarantee TRUST will indeed converge to the globally optimal value. To address this problem, Barhen and Protopopescu [14] present two strategies. The first involves augmenting the repeller term E_{rep} with a weight based on the gradient behavior of f (basically, this is the same concept as momentum in conjugate gradient descent). The effect is to guide the gradient descent search in step 3 to the closest highest ridge value on the surface of $E(x, x^*)$. The second strategy is to reduce the multi-dimensional problem into a one-dimensional problem for which TRUST is guaranteed to converge by using hyperspiral embedding from differential geometry.

TRUST is computationally efficient in the number of function evaluations made during the convergence process. Usually, the most computationally demanding aspect of global optimization is evaluating the cost function [16]. TRUST was compared to several other global optimization methods [12] and was found to need far fewer function evaluations to converge. Popular methods of eluding

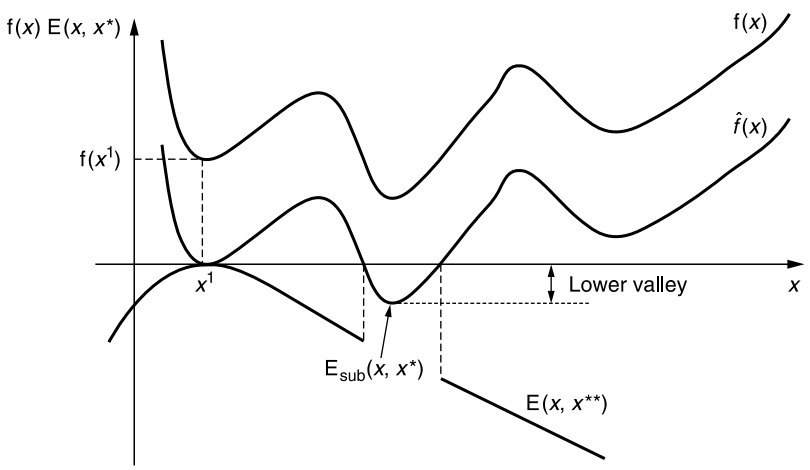


Figure 17.2. TRUST in one-dimensional case.

© 2005 by Chapman & Hall/CRC

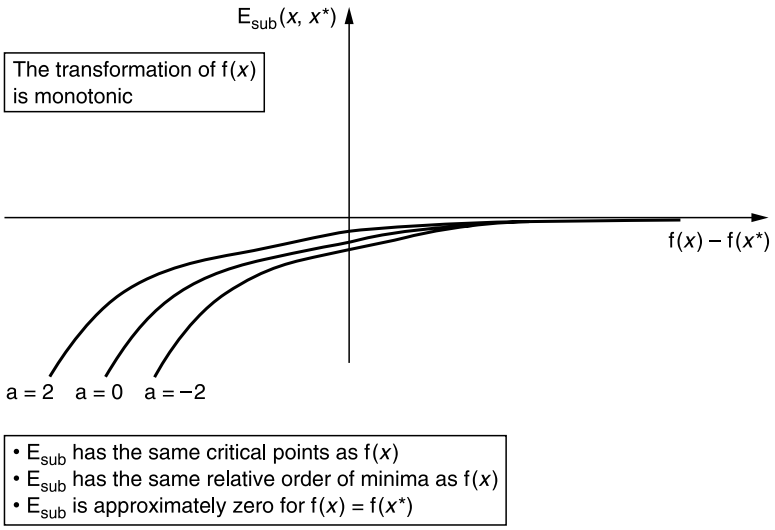


Figure 17.3. TRUST: subenergy tunneling function.

local minima basins other than conjugate gradient descent include simulated annealing and genetic algorithms. Both of these are probabilistic, since their entrapment avoidance mechanism requires randomness. TRUST, on the other hand, is deterministic and has a well-defined stopping criterion.

17.5 Tabu Search

Tabu search is one of a class of nonmonotonic search methods that have been proposed for solving optimization problems. All methods discussed have been constructed to find globally optimal solutions in problems containing local minima. Local minima in optimization problems can be either the result of the feasible set being defined by a nonconvex surface, or the function $f(x)$ being nonlinear. Optimization problems containing local minima are more difficult than the linear problems, which can be solved by linear programming.

Monotonic methods include simulated annealing and threshold acceptance. The majority of search heuristics start at a point in the parameter space and move to neighboring points whose value of $f(x)$ is inferior to the current value. Monotonic methods avoid local minima by occasionally moving to a neighboring point whose value of $f(x)$ is superior to the current value. The amount of increase that will be accepted by the method can be either probabilistic (simulated annealing) or deterministic (threshold acceptance) in nature and is dependent on a parameter whose value decreases as the search progresses.

Nonmonotonic methods also occasionally move to points in the search space where the value of $f(x)$ is superior to the current value, but the algorithm's ability to make this move is not dependent on a strictly decreasing parameter. A number of methods have been proposed, including Old Bachelor Acceptance [17] and many variations of tabu search [18]. Old Bachelor Acceptance is nonmonotone, in that it uses a parameter which is dependent on whether or not previous moves have been accepted, and on the number of moves remaining before the final answer is due [17]. Tabu search is nonmonotone, in that it disqualifies a number of moves due to the recent history of moves made by the algorithm [19]. We limit our discussion to tabu search, since it is the most widely implemented heuristic of this class.

Tabu search involves modifying an existing heuristic search by keeping a list of the nodes in the search space which were visited most recently by the search algorithm. These points then become "tabu" for the algorithm, where tabu means that these points are not revisited as long as they are on the list. This simple modification will allow a search algorithm to eventually climb out of shallow local minima in the search space. It requires less computation than simulated annealing, while providing

© 2005 by Chapman & Hall/CRC

roughly equivalent or superior results [19]. Several questions are being studied as to how to optimize tabu searches, such as the optimal length for the tabu list [20] and methods for implementing parallel searches [21].

The implementation of the tabu search we have used relies on a "greedy" heuristic. The search can move one position in each direction. The algorithm evaluates the fitness function for each of these possibilities. Naturally, the search chooses to visit the next node in the direction with the minimum value for the fitness function. When one of these parameter sets is visited it is placed on the tabu list.

Values on the tabu list are disqualified for consideration in the future. Should the search arrive at a neighboring point later, the fitness function value given to parameter sets on the tabu list is set to a very large value, essentially disqualifying it from consideration.

As each parameter set is visited by the search, the value attributed to it by the fitness function is compared with the parameter set already visited with the smallest value for the fitness function up to this point. If the value is smaller then the parameter set now becomes the best fit found.

It is impossible to find a clear stopping criterion for this algorithm, since the only way to be sure that the global minimum for the fitness function has been found is through an exhaustive search of the search space.

Tabu search has been successfully implemented in a number of systems, solving problems related to intelligent sensor processing. Chiang and Kuvelis [22] have used a variation of tabu search in order to find paths for navigation by autonomous guided vehicles. Autonomous guided vehicles are of increasing importance in materials handling and manufacturing applications. Another area of direct relevance for the sensor architectures topic is the use of distributed systems. Research has been done using tabu search to map tasks onto processors in distributed systems in a way that minimizes communication time requirements for the system [23]. The design of electromagnetic sensor devices has also been optimized through research using tabu search methods. The variables involved in designing a magnet that produces a homogeneous field can be reduced to a finite alphabet that the tabu search strategy uses to design near-optimal components for magnetic resonance imaging systems [24].

17.6 Artificial Neural Networks

Artificial neural networks, like simulated annealing and genetic algorithms, originated as a simulation of a natural phenomenon. The first artificial neural network application was developed in the 1940s by McCulloch and Pitts as a system designed to mimic a theory of how biological nervous systems work [25]. Since then, a number of variations of artificial neural networks have been developed, some of which bear little resemblance to functioning biological systems [25–28].

Neural networks are also called *connectionist learning schemes,* since they store information in weights given to connections between the artificial neurons [27]. In the vast majority of implementations, the artificial neurons are a very primitive computational device. The individual neuron takes weighted inputs from a number of sources, performs a simple function, and then produces a single output. The thresholding function is commonly a step function, a sigmoid function, or another similar function [28]. The computational power of a neural network comes from the fact that a large number of these simple devices are interconnected in parallel. The overall system may be implemented either in hardware or software.

Probably the most common architecture for neural networks is termed a *multi-layer feed-forward* network. These networks can consist of any number of neurons arranged in a series of layers. Each layer i of the network takes input from the previous layer $i-1$, and its output is used as input by layer $i+1$. Layer 0 of the network is the input data, and the outputs of the last layer n of artificial neurons is the output of the network. The layers between 0 and n are termed *hidden layers*. Since no loops exist within the network, the system can be treated as a simple black box. The number of neurons at each layer in multi-layer feed-forward networks can be any integer greater than zero, and the numbers of inputs and outputs are not necessarily equal. A neural network of this type, with n inputs and m outputs, can approximate any function that maps binary vectors of length n onto binary vectors of length m.

© 2005 by Chapman & Hall/CRC

In doing this, each neuron defines a hyperplane in the solution space. The neuron fires if the region defined by the inputs is on one side of the hyperplane, and does not fire if the region defined by the inputs is on the other side [29].

The architecture needed for a neural network depends on the number of hyperplanes needed to define the desired mapping, and whether the mapping is defined by a convex region, a nonconvex region without holes, or a nonconvex region with holes. A convex region can be defined by a network with only one layer. Nonconvex regions require two layers, and regions containing holes require three layers. Although no limit exists to the number of layers that could be used, more than three layers are not required for this type of application [28]. No clear guidelines exist for determining the number of neurons required at each level for a given application.

Each neuron at layer i outputs the value of a function of the linearly weighted outputs of all the neurons at layer $i-1$. As mentioned earlier, this function is normally a step function, a sigmoid function, or a similar function such as the hyperbolic tangent. A method is needed for determining the weights to be applied to each input. The most widely used method for doing this is *back propagation* [29]. For back propagation to be used, the function computed must be differentiable. A more computationally efficient way for computing the desired weight tables can be found in [28].

Neural networks can also be implemented to serve as self-associative memories. The network used for this type of application generally consists of a single layer of neurons, all of which are fully connected. These types of network are called *Hopfield networks.* The weights can then be defined so that the network has a number of stable states. The input to the network is used as initial values for the neurons. Each neuron then automatically readjusts its output to move towards one of the stable states. This type of learning is called *Hebbian learning* [28].

In addition to the above applications, neural networks can automatically classify inputs into equiprobable sets. This approach was first pioneered by Kohonen. These networks compute their own weights to determine rational groupings of the input vectors [25].

Probably the most appealing aspect of neural networks is that feed-forward networks can infer functions that differentiate between groups of objects based on raw data in the form of training sets. By using the network responses to data to readjust the weighting parameters, networks can be designed to infer any m to n mapping function. Unfortunately, knowing what sets of data will work adequately as training data is often difficult. Training sets that do not accurately represent the overall population of data will result in a network that is faulty. Training sets that are too large can result in *over-training* of the network so that the network does not correctly generalize the problem and only recognizes the instances used in training the network. Recently, Rao [30,31] has derived bounds on the size of the training set necessary to be certain of correctly learning a concept.

One drawback to the use of neural networks is the potential size of the system. For systems involving $n \times m$ two-dimensional images, each neuron at the lowest level would require an $n \times m$ matrix of weights to be able to treat the raw data. This is generally unfeasible. For this reason, most neural networks that treat visual images require a preprocessing step to extract features of interest. This step amounts to encoding the image into an alphabet that a neural network can easily manipulate. Often, this step is the most difficult one in developing these types of pattern recognition system.

17.7 Fuzzy Logic

The final soft-computing method we discuss is fuzzy logic and fuzzy sets. Fuzzy logic is a type of multi-valued logic that has been identified as a new technology with special promise for use with sensor fusion problems [32], as are fuzzy sets [33]. Fuzzy sets and fuzzy logic can be presented as a general theory that contains probability [34] and Dempster–Shafer evidential reasoning [35] as special cases. Fuzzy sets are similar to traditional set theory, except that every member of a fuzzy set has an associated value assigned by a *membership function* that quantifies the amount of membership the element has in the function [33].

© 2005 by Chapman & Hall/CRC

In addition to membership, a number of functions exist for fuzzy sets that are analogous to the functions performed on traditional, or *crisp*, sets. In fact, a number of possible functions exist for the operations of complementation, union, and intersection with fuzzy sets. Similarly, fuzzy logic presents a simple framework for dealing with uncertainty [34]. The number of possible implementations of fuzzy sets and fuzzy logic make it difficult saying exactly which implementation is best suited for use in sensor fusion methodology.

Sensor fusion is most closely associated with two areas in which fuzzy sets and logic have been used with success: measurement theory and the development of fuzzy controllers [34]. Both areas are closely related to the sensor fusion problems, and possibly a number of existing methods could be phrased in terms of fuzzy methodology. Other related areas that fuzzy techniques have successfully been applied to are pattern recognition and inference of statistical parameters [34]. These point to the potential for successful research applying fuzzy methods directly to sensor fusion questions. Of particular interest is the growing synergism between the use of fuzzy sets, artificial neural networks, and genetic algorithms [34].

17.8 Linear Programming

Although it is not a soft-computing method, linear programming is discussed here for completeness, since it is probably the most widely used optimization methodology. Given a cost vector $c = (c_1, c_2, \ldots, c_n)$, linear programming searches for the solution vector $x = (x_1, x_2, \ldots, x_n)$, which optimizes $f(x) = \sum c_i x_i$ and also satisfies a number of given constraints $\sum a_{ij} x_j \leq b_i$ (or $\sum a_{ij} x_j \geq b_i$). Note that the cost function optimization can solve either maximization or minimization problems, depending on the signs of the coefficients c_j. The set of problems that can be put into this form is large and it includes a number of practical applications. Note that the linear nature of this problem formulation excludes the existence of local minima of $f(x)$ in the parameter space.

Two major classes of methods exist for solving linear programming problems: the simplex method and interior methods. The simplex method was developed by a research team led by Dantzig in 1947 [36]. Conceptually, it considers the problem as an N-dimensional space where each vector x is a point in the space, and each constraint defines a hyperplane dividing the space into two half-spaces. The constraints, therefore, define a convex region containing the set of points which are possible answers to the problem, called the feasible set [37]. The set of optimal answers must contain a vertex. The simplex algorithm starts from a vertex on the N-dimensional surface and moves in the direction of greatest improvement along the surface until the optimal point is found [36,37].

The first interior method was found by Karmarkar [36]. Interior methods start with a point in the feasible set and move through the interior of the feasible set towards the optimal answer without being constrained to remain on the defining surface. Research is active in finding less computationally expensive implementations of both interior and simplex methods.

To represent how linear programming works, we solve an example problem using a variation of the simplex method. We attempt to maximize the equation $5x_1 + 3x_2 + 2x_3$ within the constraints

$$\begin{aligned} x_1 - x_2 &\leq 6 && \text{inequality 1} \\ x_1 + 3x_2 - x_3 &\leq 14 && \text{inequality 2} \\ 3x_1 + x_2 + 2x_3 &\leq 22 && \text{inequality 3} \end{aligned} \tag{17.6}$$

To make the three constraints equations instead of inequalities, we create fictitious variables: $x_{1'}$ is the difference between the left-hand side of inequality 1 (i.e., $x_1 - x_2$) and 6; similarly, $x_{2'}$ and $x_{3'}$ are the respective differences between the left-hand sides of inequalities 2 (3) and 14 (22). This definition also provides us with a starting point on the hypersurface that defines the feasible set. The starting point is $x_1 = 0$, $x_2 = 0$, $x_3 = 0$, $x_{1'} = 6$, $x_{2'} = 14$, and $x_{3'} = 22$, or in vector form (0, 0, 0, 6, 14, 22).

© 2005 by Chapman & Hall/CRC

We can now phrase the problem at this starting point in tabular form:

c_{ij}	j	x_1	x_2	x_3	$x_{1'}$	$x_{2'}$	$x_{3'}$	u
0	$x_{1'}$	1	−1	0	1	0	0	6
0	$x_{2'}$	1	3	−1	0	1	0	14
0	$x_{3'}$	3	1	2	0	0	1	22
								F
	Δ	5	3	2	0	0	0	0

Conceptually, each constraint defines a hyperplane that separates the feasible set from the rest of the answer space. If an optimal answer exists, then it will be located on at least one of the hyperplanes. It may exist at an intersection of all the hyperplanes. The left-most box in the tabular representation above contains two columns: j says which of the variables considered is the limiting factor for the hyperplane defined by the corresponding constraint, and c_{ij} is the value of one unit of j. The rightmost box u shows the number of units of j in the current answer. Beneath the right-most box is F, the value of the current answer, which is the summation of c_{ij} times the corresponding element of the right-most box, in this case zero.

The box marked Δ are the coefficients of the corresponding variables in the equation to be optimized. The middle box is simply a translation of the constraint equations into matrix form. Verify that you understand the derivation of these tables before continuing.

The algorithm follows Dantzig's simplex rules by moving the solution in the search space in the direction that produces the greatest rate of improvement. This variable has the largest coefficient in Δ. For the table above, this is variable x_1, since it has the coefficient 5. To determine which constraint will limit the growth of x_1, the value of the coefficient of x_1 for each constraint is used to divide the corresponding value in u. The smallest resulting value determines the limiting constraint. In this case the values are 6/1, 14/1, and 22/3. Since 6/1 is the smallest value, x_1 replaces $x_{1'}$ as the limiting variable in the first constraint. The new table is now:

c_{ij}	j	x_1	x_2	x_3	$x_{1'}$	$x_{2'}$	$x_{3'}$	u
5	x_1	1	−1	0	1	0	0	6
0	$x_{2'}$	0	4	−1	−1	1	0	8
0	$x_{3'}$	0	4	2	−3	0	1	4
								F
	Δ	0	8	2	−5	0	0	30

When we replace $x_{1'}$ with x_1 in constraint 1, the following steps are performed:

1. The row corresponding to constraint 1 in the left-most box is changed to contain the coefficient of the limiting variable in the column c_{ij} and to contain the limiting variable in column j.
2. The row corresponding to constraint 1 is set to *canonical form* by dividing all elements of the middle box and the corresponding element of u by the coefficient of x_1 in that row. In this case the value is 1, so the row remains unchanged.
3. The value of the coefficient of x_1 is removed from all the other rows by subtracting a multiple of row 1 from the other rows. Note that the corresponding modifications are made to u as well.
4. Similarly, the value of the coefficient of x_1 is set to 0 in Δ by subtracting a multiple of row 1 from Δ.

The value of F is recomputed using the new values in c_{ij} and u.

The variable with the largest coefficient in Δ is now x_2. Since the coefficient of x_2 is negative in the row corresponding to constraint 1, we do not consider that constraint. The two remaining constraints

© 2005 by Chapman & Hall/CRC

are constraint 2 with a value of 8/4, and constraint 3 with a value of 4/4. The value corresponding to constraint 3 is the smallest, so we replace variable $x_{3'}$ with variable x_2 as the limiting factor in constraint 3. Using the same process as above, this gives us the following table:

c_{ij}	j	x_1	x_2	x_3	$x_{1'}$	$x_{2'}$	$x_{3'}$	u
5	x_1	1	0	1/2	1/4	0	1/4	7
0	$x_{2'}$	0	0	− 3	2	1	− 1	4
3	x_2	0	1	1/2	− 3/4	0	1/4	1
								F
	Δ	0	0	− 2	1	0	− 2	38

The only positive coefficient in Δ is now that of $x_{1'}$, so it will be used to replace one of the existing coefficients. Since its coefficient is negative in constraint 3, we do not consider that constraint. The value corresponding to constraint 2 (4/2) is less than the value corresponding to constraint 3 (4*7). The same process as before is used to replace $x_{2'}$ with $x_{1'}$. This final table results:

c_{ij}	j	x_1	x_2	x_3	$x_{1'}$	$x_{2'}$	$x_{3'}$	u
5	x_1	1	0	7/8	0	− 1/8	3/8	6 1/2
0	$x_{1'}$	0	0	− 3/2	1	1/2	− 1/2	2
3	$x_{2'}$	0	1	− 5/8	0	3/8	− 1/8	2 1/2
								F
	Δ	0	0	− 1/2	0	− 1/2	− 3	40

Since all elements of Δ are now negative or zero, no improvements can be made to the current answer. The process stops and 40 is the optimal answer to our equation. The optimal answer consists of 6.5 units of x_1, 2.5 units of x_2, and no units of x_3.

17.9 Summary

This chapter provided a brief description of optimization and soft computing techniques. These techniques are widely used decision-making tools for difficult optimization problems. Each method presented can be considered a class of heuristics. Brooks and co-works [2,3] compares the use of many of these techniques for many sensor network design issues. The results of Brooks and co-works [2,3] can be used to help decide which technique is best suited for a given application.

References

[1] Brooks, R.R. *et al.*, Automatic correlation and calibration of noisy sensor readings using elite genetic algorithms, *Artificial Intelligence*, 84(1–2), 339, 1996.

[2] Brooks, R.R. *et al.*, A Comparison of GAs and simulated annealing for cost minimization in a multi-sensor system, *Optical Engineering*, 37(2), 505, 1998.

[3] Brooks, R.R. and Iyengar, S.S., *Multi-Sensor Fusion: Fundamentals and Applications with Software*, Prentice Hall PTR, Saddle River, NJ, 1998.

[4] Chen, Y. *et al.*, Efficient global optimization for image registration, *IEEE Transactions on Knowledge and Data Engineering*, 14(1), 79, 2002, http://www.computer.org/tkde/Image-processing.html.

[5] Holland, J.H., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.

[6] Kumar, A. *et al.*, Genetic-algorithm-based optimization for computer network expansion, *IEEE Transactions on Reliability*, 44(1), 63, 1995.

© 2005 by Chapman & Hall/CRC

[7] Bean, J.C., Genetic algorithms and random keys for sequencing and optimization, *ORSA Journal on Computing*, 6(2), 154, 1994.
[8] Painton, L. and Campbell, J., Genetic algorithms in optimization of system reliability, *IEEE Transactions on Reliability*, 14(2), 172, 1995.
[9] Van Laarhoven, P.J.M. and Aarts, E.H.L., *Simulated Annealing: Theory and Applications*, D. Reidel Publishing Co., Dordrecht, 1987.
[10] Press, W. *et al.*, *Numerical Recipes in Fortran*, Cambridge University Press, 436, 1986.
[11] De Bonet, J.S. *et al.*, @IC: finding optima by estimating probability densities, *Advances in Neural Information Processing Systems*, MIT Press, Cambridge, MA, 1997.
[12] Barhen, J. *et al.*, TRUST: a deterministic algorithm for global optimization, *Science*, 276, 16 May, 1094, 1997.
[13] Cetin, B.C. *et al.*, Teminal repeller unconstrained subenergy tunneling (TRUST) for fast global optimization, *Journal of Optimization Theory and Application*s, 77(l), 97, 1993.
[14] Barhen, J. and Protopopescu, V., Generalized TRUST algorithms for global optimization, in *State of the Art in Global Optimization*, Floudas, C.A. and Pardalos, P.M. (eds), Kluwer Acedemic Publishers, 1996, 163.
[15] Levy, A.V. and Montalvo, A., The tunnelling algorithm for the global minimization of functions, *SIAM Journal on Scientific and Statistical Computing*, 6, 15, 1985.
[16] Srinivas, M. and Patnaik, L.M., Genetic Algorithms: A Survey, *IEEE Computer*, 27(6), 17, 1994.
[17] Hu, T.C. *et al.*, Old Bachelor Acceptance: a new class of non-monotone threshold accepting methods, *ORSA Journal on Computing*, 7(4), 417, 1995.
[18] Glover, F., Tabu search – part 1, *ORSA Journal on Computing* 1, 190, 1989.
[19] Glover, F., Tabu thresholding: improved search by nonmonotonic techniques, *ORSA Journal on Computing*, 7(4), 426, 1995.
[20] Battiti, R. and Tecchioli, G., The reactive tabu search *ORSA, Journal on Computing* 6(2), 126, 1994.
[21] Taillard, E., Parallel taboo search techniques for the job shop scheduling problem, *ORSA Journal on Computing*, 6(2), 108, 1994.
[22] Chiang, W.C. and Kouvelis, P., Simulated annealing and tabu search approaches for unidirectional flowpath design for automated guided vehicle systems, *Annals of Operation Research*, 50, 1994.
[23] Chakrapani, J. and Skorin-Kapov, J., *Mapping Tasks to Processors to Minimize Communication Time in a Multiprocessor System, The Impact of Emerging Technologies on Computer Science and Operations Research*, Kluwer Academic Publishers, 1995.
[24] Fanni, A. *et al.*, Tabu search for continuous optimization of electromagnetic structures, in *International Workshop on Optimization and Inverse Problems in Electromagnetism*, June, 1996.
[25] Davalo, E. and Daïm, P., *Des Reseaux de Neurones*, Editions Eyrolles, Paris, 1989.
[26] Gulati, S. *et al.*, Neurocomputing formalisms for computational learning and machine intelligence, in *Advances in Computers*, *33,* Gulati, S., Barhen, J. and Iyengar, S.S. (eds), Academic Press, Boston, 1991.
[27] Hinton, G.E., Connectionist Learning Procedures, in *Machine Learning*, Carbonnell, J. (ed.), MIT/Elsevier, 185, 1992.
[28] Kak, S., *Neural Networks, Iterative Maps, and Chaos*. Course Notes, Louisiana State University, 1994.
[29] Rojas, R., *Neural Networks: A Systematic Introduction*, Springer Verlag, Berlin, 1996.
[30] Rao, N.S.V., Fusion rule estimation in multiple sensor systems with unknown noise distributions, in *Parallel and Distributed Signal and Image Integration Problems*, Madan, R.N. *et al.*, World Scientific, Singapore, 263, 1995.
[31] Rao, N.S.V., Multiple sensor fusion under unknown distributions, in *Proceedings of Workshop on Foundations of Information/Decision Fusion with Applications to Engineering Problems*, Rao *et al.*, 174, 1996.
[32] Luo, R. and Kay, M., Data fusion and sensor integration: State-of-the-art 1990s, in *Data Fusion in Robotics and Machine Intelligence*, Abidi, M.A. and Gonzales, R.C. (eds). Academic Press, Boston, 1992, 7.

© 2005 by Chapman & Hall/CRC

[33] Waltz, E.L. and Llinas, J., *Sensor Fusion*, Artech House, Norwood, MA, 1991.
[34] Klir, G.J. and Yuan, B., *Fuzzy Sets and Fuzzy Logic: Theory and Applications*, Prentice Hall, Englewood Cliffs, NJ, 1995.
[35] Dubois, D. and Prade, H., Combination of fuzzy information in the framework of possibility theory, *Data Fusion in Robotics and Machine Intelligence*, Abidi, M.A. and Gonzales, R.C. (eds.), Academic Press, Boston, 481, 1992.
[36] Alj, A. and Faure, R., *Guide de la Recherche Operationelle*, vol. 1–2, Masson, Paris, 1990.
[37] Strang, G., *Linear Algebra and Its Applications*, Academic Press, New York, 1976.

© 2005 by Chapman & Hall/CRC

18

Estimation and Kalman Filters

David L. Hall

18.1 Introduction

Within the overall context of multi-sensor data fusion, Kalman filters provide a classic sequential estimation approach for fusion of kinematic and attribute parameters to characterize the location, velocity and attributes of individual entities (e.g. targets, platforms, events, or activities). This chapter (based extensively on material originally presented in chapter 4 of [1]) provides an introduction to estimation, sequential processing, and Kalman filters. The problem of fusing multi-sensor parametric data (from one or more sensors) to yield an improved estimate of the state of the entity is a classic problem. Examples of estimation problems include:

1. Using positional data such as line-of-bearing (angles), range or range-rate observations to determine the location of a stationary entity (e.g. determining the location and velocity of a ground-based target using a distributed network of ground sensors).
2. Combining positional data from multiple sensors to determine the position and velocity of a moving object as a function of time (the tracking problem).
3. Estimating attributes of an entity, such as size or shape, based on observational data.
4. Estimating the parameters of a model (e.g. the coefficients of a polynomial), which represents or describes observational data.

The estimation problem involves finding the value of a state vector (e.g. position, velocity, polynomial coefficients) that best fits, in a defined mathematical sense, the observational data. From a mathematical viewpoint, we have a redundant set of observations (viz. more than the minimum number of observations for a minimum data-set solution), and we seek to find the value of set of parameters that provides a "best fit" to the observational data. In general, the observational data are corrupted by measurement errors, signal propagation noise, and other factors, and we may or may not know *a priori* the statistical distribution of these sources of error. Figure 18.1 shows a sample target-tracking problem in which multiple targets are tracked by multiple sensors.

In this example, it is easy to determine which observations "belong" to which target tracks (because of our artificial representation of data using geometrical icons). However, in general, we do not know

© 2005 by Chapman & Hall/CRC

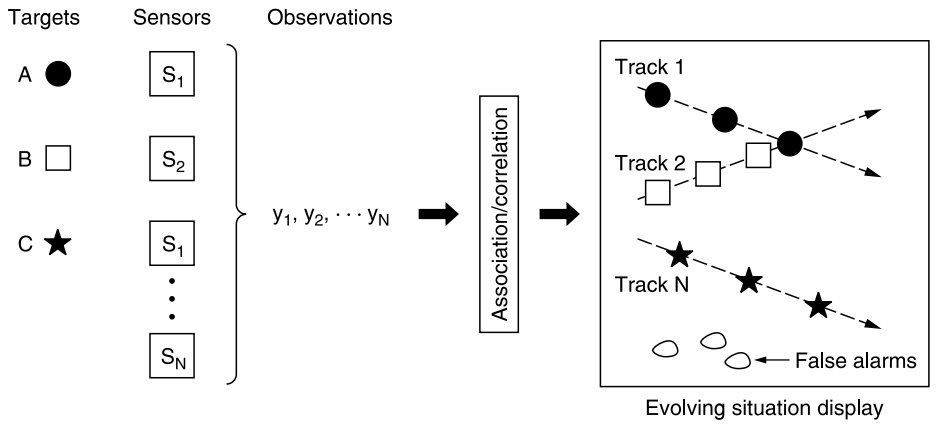


Figure 18.1. Conceptual multi-target, multi-sensor target-tracking problem[2]. Data from multiple sensors, observing multiple targets, are fused to obtain estimates of the position, velocity, attributes, and identity of the targets.

a priori how many targets exist, which observations belong to which targets, or whether observations are evidence of a moving target or are simply false alarms.

Estimation problems may be dynamic, in which the state vector changes as a function of time, or static, in which the state vector is constant in time. This chapter introduces the estimation problem and develops specific solution strategies. For simplicity, it is assumed that the related problems of data registration and data correlation have been (or can be) solved. That is, we assume that the observational data are allocated to distinct groups or sets, each group belonging to a unique entity or object. In practice, data association and estimation techniques must be interleaved to develop an overall solution, especially for multi-target tracking problems. A detailed discussion of techniques for data correlation and association is provided by Hall [2].

The history of estimation techniques has been summarized by Sorenson [3]. The first significant effort to address estimation was Karl Friedrick Gauss's invention of the method of least squares to determine the orbits of planets, asteroids, and comets from redundant data sets. In celestial mechanics, techniques for determining orbital elements from minimum data sets are termed initial orbit methods or minimum data-set techniques. Gauss utilized the method of least squares in 1795 and published a description of the technique in 1809 [4]. Independently, Legendre invented the least-squares method and published his results in 1806 [5]. The resulting controversy of intellectual propriety prompted Legendre to write to Gauss complaining, "Gauss, who was already so rich in discoveries, might have had the decency not to appropriate the method of least-squares" (quoted in Sorenson [3] and Bell [6]). Gauss's contribution included not only the invention of the least-squares method, but also the introduction of such modern notions as:

1. *Observability* — the issue of how many and what types of observations are necessary to develop an estimate of the state vector.
2. *Dynamic modeling* — the need for accurate equations of motion to describe the evaluation of a state vector in time.
3. *A priori estimate* — the role of an initial (or starting) value of the state vector in order to obtain a solution.
4. *Observation* noise — set the stage for a probabilistic interpretation of observational noise.

Subsequent historical developments of estimation techniques include Fisher's probabilistic interpretation of the least-squares method [7] and definition of the maximum likelihood method, Wiener's [8] and Kolmogorov's [9] development of the linear minimum mean-square-error method, and Kalman's formulation of a discrete-time, recursive, minimum mean-square filtering technique

© 2005 by Chapman & Hall/CRC

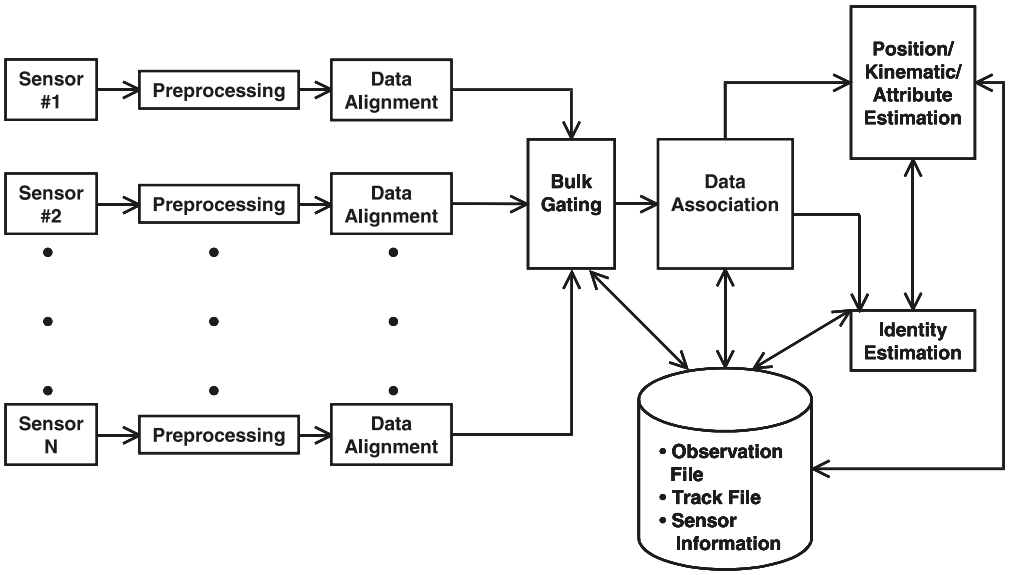


Figure 18.2. A conceptual view of the estimation process [2]. Data from each sensor or source must be preprocessed to align the data with respect to a common coordinate frame and grouped (associated/correlated) such that each data set represents data belonging to an individual target. Positional, kinematic and attribute estimation can be performed by various techniques including Kalman filtering.

(viz. the Kalman filter [10]). The Kalman filter (also independently described by Bucy and Sterling) was motivated by the need for rapid prediction of the position of early spacecraft using very limited computational capability.

Numerous texts and papers have been published on the topic of estimation (and in particular on sequential estimation). Blackman [11] provides a detailed description of sequential estimation for target tracking, Gelb [12] describes sequential estimation from the viewpoint of control theory, and Ramachandra [13] describes the application of Kalman filtering for radar tracking. In addition, Grewal and Andrews [14] provide practical advice and MATLAB code for implementing Kalman filters.

A conceptual view of the general estimation processing flow is illustrated in Figure 18.2. The situation is illustrated for a positional fusion (e.g. target tracking and identification) problem. A number of sensors observe location parameters, such as azimuth, elevation, range, or range rate, and attribute-parameters such as radar cross-section. The location parameters may be related to the dynamic position and velocity of an entity via observation equations. For each sensor, a data alignment function transforms the "raw" sensor observations into a standard set of units and coordinate reference frame. An association/correlation process groups observations into meaningful groups — each group representing observations of a single physical entity or event. The associated observations represent collections of observation-to-observation pairs, or observation-to-track pairs, which "belong" together. An estimation process combines the observations to obtain a new or improved estimate of a state vector, $\boldsymbol{x}(t)$, which best fits the observed data. The estimation problem illustrated in Figure 18.2 is the level-1 process within the Joint Directors of Laboratories data fusion process model [1,15,16]. It also assumes a centralized architecture in which observations are input to an estimation process for combination. In practice, other architectures such as distributed processing or hybrid-processing could be used for the estimation fusion process.

18.2 Overview of Estimation Techniques

Estimation techniques have a rich and extensive history. An enormous amount has been written, and numerous methods have been devised for estimation. This section provides a brief overview of the

© 2005 by Chapman & Hall/CRC

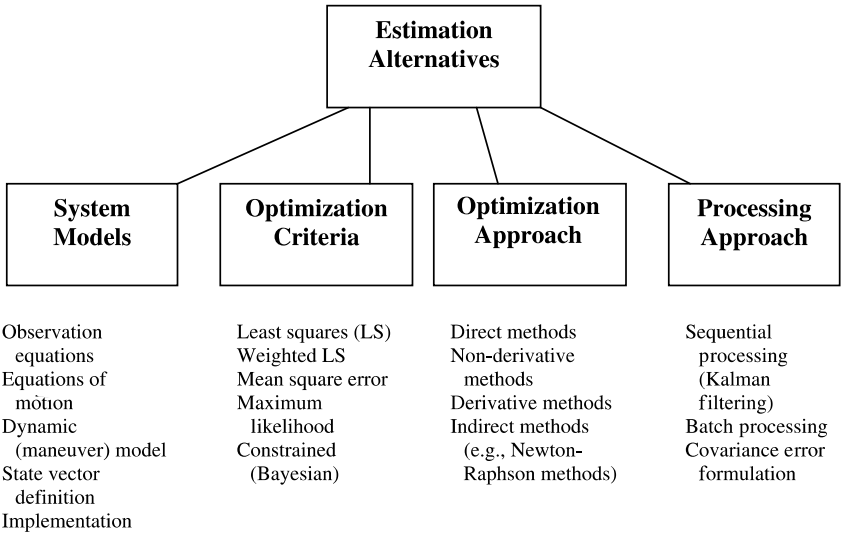


Figure 18.3. Overview of estimation alternatives. Design of a process for state vector estimation requires selection of system models, optimization criteria, an optimization approach, and a basic data processing approach.

choices and common techniques available for estimation. Figure 18.3 summarizes the alternatives and issues related to estimation. These include:

1. *System models*. What models will be selected to define the problem under consideration? What is to be estimated (viz. what is the state vector sought)? That is, what set of parameters are sufficient and necessary to provide a description of the system "state"? How do we predict the state vector in time? How are the observations related to the state vector? What assumptions (if any) can we make about the observation process (e.g. noise, biases, etc.)?
2. *Optimization Criteria*. How will we define a criteria to specify *best fit*? That is, what equation will be used to specify that a state vector best fits a set of observations?
3. *Optimization Approach*. Having defined a criterion for *best fit*, what method will be used to find the unknown value of the state vector which satisfies the criterion?
4. *Processing Approach*. Fundamentally, how will the observations be processed, e.g. on a batch mode in which all observations are utilized after they have been received, or sequentially, in which observations are processed one at a time as they are received?

18.2.1 System Models

An estimation problem is defined by specifying the state vector, observation equations, equations of motion (for dynamic problems), and other choices, such as data editing, convergence criteria, and coordinate systems, that are necessary to specify the estimation problem. We will address each of these in turn.

A fundamental choice in estimation is to specify what parameters are to be estimated, i.e. what is the independent variable or state vector $\mathbf{x}(t)$ whose value is sought? For positional estimation, a typical choice for $\boldsymbol{x}$ is the coordinates necessary to locate a target or entity. Examples include the geodetic latitude and longitude (ϕ, λ) of an object on the surface of the Earth, the three-dimensional Cartesian coordinates (x, y, z) of an object with respect to Earth-centered inertial coordinates, or the range and angular direction (r, azimuth, elevation) of an object with respect to a sensor. For nonpositional estimation, the state vector may be selected as model coefficients (e.g. polynomial coefficients) to represent or characterize data. State vectors may also include coefficients that model sensor biases and

© 2005 by Chapman & Hall/CRC

basic system parameters. For example, at the National Aeronautics and Space Administration (NASA) Goddard Space Flight Center, large-scale computations are performed to estimate a state vector having several hundred components, including; position and velocity of a spacecraft, spherical harmonic coefficients of the Earth's geo-potential, sensor biases, coefficients of atmospheric drag, precise locations of sensors, and many other parameters [17].

The choice of what to estimate depends upon defining what parameters are necessary to characterize a system and to determine the future state of a system, and what parameters may be estimated based on the observed data. The latter issue is termed *observability*. The issue concerns the extent to which it is feasible to determine components of a state vector based on observed data. A weak relationship may exist between a state vector element and the data. Alternatively, two or more components of a state vector may be highly correlated, with the result that variation of one component to fit observed data may be indistinguishable from variation in the related state vector component. Hall and Waligora [18] provide an example of the inability to distinguish between camera biases and orientation (attitude) of Landsat satellite data. Deutsch [19] also provides a discussion of observability. The general rule of thumb in selecting a state vector is to choose the minimum set of components necessary to characterize a system under consideration. It is often tempting to choose more, rather than fewer, components of a state vector. An example of this occurs when researchers attempt to represent observational data using a high-order polynomial. Hence, a cautious, *less is better* approach is recommended for selecting the components of a state vector.

A second choice required to define the estimation problem is the specification of the observation equations. These equations relate the unknown state vector to predicted observations. Thus, if $\boldsymbol{x}(t)$ is a state vector, and $\mathbf{y}_i(t_i)$ is an observation, then

$$\mathbf{z}_i = g(x_i(t_i)) + \eta \tag{18.1}$$

predicts an observation, $\mathbf{z}_i(t_i)$, which would match $\mathbf{y}_i(t_i)$ exactly if we knew the value of $\boldsymbol{x}$ and we also knew the value of the observational noise η. The function $g(\mathbf{x}_i(t_i))$ represents the coordinate transformations necessary to predict an observation based on an assumed value of a state vector. If the state vector $\boldsymbol{x}$ varies in time, then the estimation problem is dynamic and requires further specification of an equation of motion which propagates the state vector at time t_0 to the time of an observation t_i, i.e.:

$$\boldsymbol{x}(t_i) = \Phi(t_i, t_0)\boldsymbol{x}(t_0) \tag{18.2}$$

The propagation of the state vector in time [Equation (18.2)] may involve a simple truncated Taylor series expansion

$$\boldsymbol{x}(t_0) = \boldsymbol{x}(t_0) + \dot{\boldsymbol{x}}(t_0)\Delta t + \frac{1}{2}\ddot{\boldsymbol{x}}(t_0)\Delta t^2 \tag{18.3}$$

where $\Delta t = t_i - t_0$, and $\dot{\boldsymbol{x}}$ represents the velocity at time t_0 and $\ddot{\boldsymbol{x}}$ represents the acceleration at time t_0. In other situations, more complex equations of motion may be required. For example, in astro-dynamical problems, the equation of motion may involve second-order, nonlinear, simultaneous differential equations in which the acceleration depends upon the position, velocity, and orientation of a body and the positions of third bodies such as the sun and moon. In that case, the solution of the differential equations of motion requires significant computational effort utilizing numerical integration techniques. Examples of such a problem may be found in Zavaleta and Smith [17] and Deutsch [19]. The selection of the equations of motion, Equation (18.2), depends on the physics underlying the dynamic problem. Tracking problems may require models to predict propulsion, target

© 2005 by Chapman & Hall/CRC

maneuvering, motion over terrain or through surrounding media, or even move–stop–move motions. Selection of an appropriate equation of motion must trade off physical realism and accuracy versus computational resources and the required prediction interval. For observations closely spaced in time (e.g. for a radar tracking an object for a brief interval) a linear model may be sufficient. Otherwise, more complex (and computationally expensive) models must be used. A special difficulty for positional estimators involves maneuvering targets. Both Blackman [11] and Waltz and Llinas [20] discuss these issues.

Closely coupled with the selection of the equations of motion is the choice of coordinate systems in which the prediction is performed. Some coordinate reference frames may be *natural* for defining an equation of motion. For example, Earth-centered (geocentric) inertial Cartesian coordinates provide an especially simple formulation of the equations which describe the motion of a satellite about the Earth [19,21]. By contrast, for the same case, topocentric (Earth surface) noninertial coordinates provide an equation of motion that must introduce artificial acceleration components (viz. due to Coriolis forces) to explain the same motion. However, despite this, the use of a topocentric noninertial coordinate frame may be advisable from a system viewpoint. Kamen and Sastry [22] provide an example of such a tradeoff. Sanza *et al.* [23] present the equations of motion and sequential estimation equations for tracking an object in a spherical (r, θ, φ) coordinate reference frame.

18.2.2 Optimization Criteria

Having established the observation equations that relate a state vector to predicted observations, and equations of motion (for dynamic problems), a key issue involves the definition of *best fit*. We seek to determine a value of a state vector $\boldsymbol{x}(t)$ which best fits the observed data. There are several ways to define *best fit*. Each of these formulations involves a function of the residuals.

$$\boldsymbol{v}_i = [\mathbf{y}_i(t_i) - \mathbf{z}_i(t_i)] \tag{18.4}$$

Here, $\boldsymbol{v}_i$ is the vector difference between the ith observation, $\mathbf{y}_i(t_i)$, at time t_i and the predicted observation, $\mathbf{z}_i(t_i)$. The predicted observation z_i is a function of the state vector $\boldsymbol{x}(t_0)$ via Equation (18.1), and hence $\boldsymbol{v}_i$ is also a function of $\boldsymbol{x}(t_0)$.

Various functions of $\boldsymbol{v}_i$ have been defined and used for estimation. A summary of some of these criteria is provided in Table 18.1. A function is chosen which provides a measure of the extent to which the predicted observations match the actual observations. This function of the unknown state vector $\boldsymbol{x}$ is sometimes termed a loss function because it provides a measure of the penalty (i.e. poor data fit) for an incorrect estimate of $\boldsymbol{x}$. The state vector $\boldsymbol{x}$ is varied until the loss function is either a minimum or maximum, as appropriate. The solution of the estimation problem then becomes an optimization problem. Such optimization problems have been treated by many texts (see for example Wilde and Beightler [24]).

Perhaps the most familiar definition of best fit are the LS and WLS formulations. The WLS expression in vector form may be written as

$$L(\boldsymbol{x}) = \boldsymbol{v} W \boldsymbol{v}^{\mathrm{T}} \tag{18.5}$$

Equivalently:

$$L(\boldsymbol{x}) = [(\mathbf{y}_i - \mathbf{z}_i), \ldots, (\mathbf{y}_n - \mathbf{z}_n)] \begin{pmatrix} w_{11} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & w_{nm} \end{pmatrix} \begin{bmatrix} (\mathbf{y}_1 - \mathbf{z}_1) \\ \vdots \\ (\mathbf{y}_n - \mathbf{z}_n) \end{bmatrix} \tag{18.6}$$

© 2005 by Chapman & Hall/CRC

Table 18.1. Eamples of optimization criteria

Criteria	Description	Mathematical Formulation	Comments
Least squares (LS)	Minimize the sum of the squares of the residuals	$L(\boldsymbol{x}) = vv^{\mathrm{T}}$	Earliest formulation proved by Gauss — no *a priori* knowledge assumed
Weighted least squares (WLS)	Minimize the sum of the weighted squares of the residuals	$L(\boldsymbol{x}) = vwv^{\mathrm{T}}$	Yields identical results to MLE when noise is Gaussian and weight matrix equals inverse covariance matrix
Mean squares error (MSE)	Minimize the expected value of the squared error	$L(\boldsymbol{x}) = \int (\mathbf{x} - \bar{\mathbf{x}}) w (\mathbf{x} - \bar{\mathbf{x}}) p(\mathbf{x}/\mathbf{y}) (\mathbf{x} - \bar{\mathbf{x}})^{\mathrm{T}} \mathrm{d}\mathbf{x}$	Minimum covariance solution
Bayesian weighted least squares (BWLS)	Minimize the sum of the weighted squares of the residuals constrained by *a priori* knowledge of $\boldsymbol{x}$	$L(\boldsymbol{x}) = \mathbf{v}w\mathbf{v}(\mathbf{x} - \mathbf{x}_0) P_{\Delta x_0} (\mathbf{x} - \mathbf{x})^{\mathrm{T}}$	Constrains the solution of $\boldsymbol{x}$ to a reasonable value close to the *a priori* estimate of $\boldsymbol{x}$
Maximum likelihood estimate (MLE)	Maximize the multivariate probability distribution function	$L(\boldsymbol{x}) = \prod_{i=1}^{n} l_1(n_1/\mathbf{x}_1) l_2(n_2/\mathbf{x}_2) \ldots l_n(n_n/\mathbf{x})$	Allows specification of the probability distribution for the noise process

© 2005 by Chapman & Hall/CRC

or

$$L(\boldsymbol{x}) = \sum_{i=1}^{n} (\mathbf{y}_i - \mathbf{z}_i) w_{ij} (\mathbf{y}_i - \mathbf{z}_i) \tag{18.7}$$

The loss function $L(\boldsymbol{x})$ is a scalar function of $\boldsymbol{x}$ which is the sum of the squares of the observation residuals weighted by W. The LS or WLS criterion is used when there is no basis to assign probabilities to $\mathbf{x}$ and $\mathbf{y}$, and there is limited information about the measurement errors. A special case involves linear least-squares, in which the predicted observations are a linear function of the state vector. In that case, $L(\boldsymbol{x})$ may be solved explicitly with a closed-form solution (see Press *et al.* [25] for the formulation).

A variation of the WLS objective function is the constrained loss function:

$$L(\boldsymbol{x}) = \boldsymbol{v} W \boldsymbol{v}^{\mathrm{T}} + (\mathbf{x} - \mathbf{x}_0) P_{\Delta x_0} (\mathbf{x} - \mathbf{x}_0)^{\mathrm{T}} \tag{18.8}$$

This expression, sometimes termed the BWLS criterion, constrains the WLS solution for $\mathbf{x}$ to be *close to* an *a priori* value of $\mathbf{x}$ (i.e. x_0). In Equation (18.8), the quantity $P_{\Delta x_0}$ represents an estimate of the covariance of x given by the symmetric matrix.

$$P_{\Delta x_0} = \begin{pmatrix} \sigma_{x1}^2 & \cdots & \sigma_{x1}\sigma_{xn} \\ & \ddots & \\ \sigma_{x1}\sigma_{xn} & & \sigma_{xn}^2 \end{pmatrix} \tag{18.9}$$

If the components of x are statistically independent, then $P_{\Delta x_0}$ is a diagonal matrix. The Bayesian criterion is used when there is prior knowledge about the value of $\mathbf{x}$ and *a priori* knowledge of associated uncertainty via $P_{\Delta x_0}$. The resulting optimal solution for $\mathbf{x}$ lies nearby the *a priori* value x_0.

The MSE formulation minimizes the expected (mean) value of the squared error, i.e. minimize

$$L(\boldsymbol{x}) = \int (\mathbf{x} - \hat{\mathbf{x}})^{\mathrm{T}} W (\mathbf{x} - \hat{\mathbf{x}}) P(\mathbf{x}|\,\mathbf{y})\, \mathrm{d}\mathbf{x} \tag{18.10}$$

$P(\mathbf{x}|\,\mathbf{y})$ is the conditional probability of state vector $\boldsymbol{x}$ given the observations $\mathbf{y}$. This formulation assumes that $\mathbf{x}$ and $\mathbf{y}$ are jointly distributed random variables. The quantity $\mathbf{x}$ is the conditional expectation of $\mathbf{x}$:

$$\hat{\boldsymbol{x}} = \int x P(x|\,y)\, \mathrm{d}x \tag{18.11}$$

The solution for $\mathbf{x}$ yields the minimum covariance of $\mathbf{x}$.

The final optimization criterion shown in Table 18.1 to define *best fit* is the maximum likelihood criterion:

$$L(\boldsymbol{x}) = \prod_{i=1}^{n} l_i(n_i/\mathbf{x}) \tag{18.12}$$

$L(\boldsymbol{x})$ is the multivariate probability distribution to model the observational noise n_i. The function $L(\boldsymbol{x})$ is the conditional probability that the observational noise at times $t_0, t_1, \ldots, t_i$ will have the values $n_0, n_1, \ldots, n_i$, if $\mathbf{x}$ is the actual or true value of the state vector. The maximum likelihood criterion selects the value of $\mathbf{x}$ which maximizes the multivariate probability of the observational noise.

If the measurement errors n_i are normally distributed about a zero mean, then $l_i(n_i/\mathbf{x})$ is given by

$$l_i(n_i/\mathbf{x}) = \frac{1}{(2\pi)^{m/2} \left| M_i^{1/2} \right|} \exp\left(-\frac{1}{2} n_i^{\mathrm{T}} M^{-1} n_i \right) \tag{18.13}$$

© 2005 by Chapman & Hall/CRC

The quantity m refers to the number of components, at each time t_i, of the observation vector and M_i is the variance of the observation at time t_i. The maximum likelihood criterion allows us to postulate non-Gaussian distributions for the noise statistics.

Selection of an optimization criterion from among the choices shown in Table 18.1 depends upon the *a priori* knowledge about the observational process. Clearly, selection of the MLE criterion presumes that the probability distributions of the observational noise are known. Similarly, the MSE criterion presumes knowledge of a conditional probability function, while the BWLS assumes *a priori* knowledge of the variance of the state vector. Under the following restricted conditions, the use of these criteria result in an identical solution for **x**. The conditions are:

1. The measurement (observational) errors are Gaussian distributed about a zero mean.
2. The errors n_i at time t_i are stochastically independent of the errors n_j, at time t_j.
3. The weight for the WLS criterion is the inverse covariance of $\boldsymbol{x}$.

Under these conditions, the WLS solution is identical to the MLE, MSE and BWLS solutions.

18.2.3 Optimization Approach

Solution of the optimization criterion to determine an estimate of the state vector $\boldsymbol{x}$ may be performed by one of several techniques. Several texts present detailed algorithms for optimization, e.g. [24–27]. Press *et al.* [25] and Shoup [27] provide computer codes to solve the optimization problem. In this section, we will provide an overview of optimization approaches and give additional detail in Sections 18.3 and 18.4 for batch and sequential estimation respectively.

Optimization techniques may be categorized into two broad classes, as illustrated in Table 18.2. Direct methods treat the optimization criteria, without modification, seeking to determine the value of $\boldsymbol{x}$ which finds an extremum (i.e. minimum or maximum) of the optimization criterion. Geometrically, direct methods are hill-climbing (or valley-seeking) techniques which seek to find the value of **x** for which $L(\boldsymbol{x})$ is a maximum or minimum. By contrast, indirect methods seek to solve the simultaneous nonlinear equation given by

$$\frac{\partial L(\boldsymbol{x})}{\partial \mathbf{x}} = \begin{bmatrix} \dfrac{\partial L}{\partial x_1} \\ \vdots \\ \dfrac{\partial L}{\partial x_m} \end{bmatrix} \tag{18.14}$$

where m is the number of components of the state vector. Indirect methods require that the optimization criterion be explicitly differentiated with respect to **x**. The problem is transformed from

Table 18.2. Categories of optimization techniques

Category	Optimization technique	Description
Direct methods	Non-derivative methods	Direct methods find the value of $\boldsymbol{x}$ that satisfies the optimization criteria (i.e. find $\boldsymbol{x}$ such that the loss function is either a minimum or maximum).
	Derivative methods	Techniques fall into two classes: derivative methods require knowledge of derivative of loss function with respect to $\boldsymbol{x}$, and nonderivative methods require only the ability to compute the loss function
Indirect methods	Newton–Raphson methods	Indirect methods find the roots of a system of equations involving partial derivatives of the loss function with respect to the state vector $\boldsymbol{x}$; i.e. the partial derivative of $L(\boldsymbol{x})$ with respect to $\boldsymbol{x}$ set equal to zero. The only successful techniques are multi-dimensional Newton–Raphson methods

© 2005 by Chapman & Hall/CRC

Table 18.3. Summary of direct methods for optimization

Type of method	Class of techniques	Algorithm/strategy	References
Nonderivative methods (do not require derivative of $L(\boldsymbol{x})$)	Downhill simplex techniques	(polygonal figure) to map out and bracket extreme of $L(\boldsymbol{x})$	[25]
	Direction set methods	Along preferred coordinate directions; examples include conjunctive direction techniques and Powell's method	[25]
Derivative methods (requires derivative of $L(\boldsymbol{x})$)	Conjunctive gradient methods	Derivative (gradient information to seek an extremum. techniques include Flecher–Reeves and Polak–Ribiere methods	[25]
	Variable metric (quasi-Newton techniques)	Dimensional Newton's method. Techniques include Davidson–Flecher–Powell and Broyden–Flecher–Goldfarb–Shannon methods	[25]

finding the maximum (or minimum) of a nonlinear equation, to one of finding the roots of m simultaneously nonlinear equations given by Equation (18.14).

A summary of techniques for a direct solution of the optimization problem is given in Table 18.3. An excellent reference with detailed algorithms and computer program listings is Press *et al.* [25]. The direct techniques may be subdivided into nonderivative techniques (i.e. those methods that require on the ability to compute $L(\boldsymbol{x})$, and derivative techniques which rely on the ability to compute $L(\boldsymbol{x})$ as well as derivatives of $L(\boldsymbol{x})$. Nonderivative techniques include simplex methods, such as that described by Nelder and Mead [28], and direction set methods, which seek successive minimization along preferred coordinate directions. Specific techniques include conjugate direction methods and Powell's methods [25,26].

Derivative methods for direct optimization utilize first- or higher-order derivatives of $L(\boldsymbol{x})$ to seek an optimum. Specific methods include conjugate gradient methods, such as the Fletcher–Reeves method and the Polak–Ribiere method [25,29]. Variable metric methods utilize a generalization of the one-dimensional Newton approach. Effective techniques include the Davidson–Flecher–Powell method and the Broyden–Flecher–Goldfarb–Shannon method. A well-known, but relatively ineffective, gradient technique is the method of steepest descent. Press *et al.* [22] provide a discussion of the tradeoffs and relative performance of these methods.

Press *et al.* [25] point out that there is only one effective technique for finding the roots of Equation (18.14), namely the multiple dimensional Newton–Raphson method. For a one-dimensional state vector, the technique may be summarized as follows. We seek to find $\boldsymbol{x}$ such that

$$\frac{\partial L(\boldsymbol{x})}{\partial \mathbf{x}} = f(\mathbf{x}) = 0 \tag{18.15}$$

Expand $f(\mathbf{x})$ in a Taylor series:

$$f(\mathbf{x} + \delta\mathbf{x}) = f(\mathbf{x}) + \frac{\partial f}{\partial \mathbf{x}}\delta\mathbf{x} + \left(\frac{\partial^2 f}{\partial \mathbf{x}^2}\right)\delta\mathbf{x}^2 + \cdots \tag{18.16}$$

Neglecting second- and higher-order terms yields a linear equation:

$$f(\mathbf{x} + \delta\mathbf{x}) = f(\mathbf{x}) + \frac{\partial f}{\partial \mathbf{x}}\delta\mathbf{x} \tag{18.17}$$

© 2005 by Chapman & Hall/CRC

To find the root of (18.17), set $f(\mathbf{x}+\delta\mathbf{x})$ equal to zero and solve for $\delta\mathbf{x}$.

$$\delta\mathbf{x} = f(\mathbf{x}) - \left(\frac{f(\mathbf{x})}{\mathrm{d}f/\mathrm{d}\mathbf{x}}\right) \tag{18.18}$$

In order to find the root of Equation (18.15), we begin with an initial value of $\mathbf{x}$, say $\mathbf{x}_i$. An improved value for x_i is given by

$$\mathbf{x}_i = \mathbf{x}_i + \delta\mathbf{x}_i \tag{18.19}$$

Equation (18.19) is solved iteratively until $\delta\mathbf{x}_i < \boldsymbol{\varepsilon}$, where ε is an arbitrarily small convergence criterion. A multi-dimensional description of the Newton–Raphson technique is described by Press *et al.* [25].

18.2.4 Processing Approach

In introducing the estimation problem and alternative design choices, we have implicitly ignored the fundamental question of how data will be processed. As illustrated in Figure 18.3, there are two basic alternatives: (1) batch processing and (2) sequential processing. Batch processing assumes that all data are available to be considered simultaneously. That is, we assume that all n observations are available, select an optimization criterion, and proceed to find the value of $\mathbf{x}$ which best fits the n observations (via one of the techniques described in Section 18.3). The batch approach is commonly used in modeling or curve-fitting problems. Batch estimation is often used in situations in which there is no time-critical element involved. For example, estimating the heliocentric orbit of a new comet or asteroid involves observations over a period of several days with subsequent analysis of the data to establish an ephemeris. Another example would entail modeling in which various functions (e.g. polynomials, log-linear functions, etc.) are used to describe data. Batch approaches have a number of advantages, particularly for situations in which there may be difficulty in association. At least in principle, one approach to find an optimal association of observations-to-tracks or observations-to-observations is simply to exhaustively try all $\{n(n-1)/2\}$ combinations. While in practice such exhaustive techniques are not used, batch estimation techniques have more flexibility in such approaches than do sequential techniques. Section 18.3 provides more detail on batch estimation, including a processing flow and discussion of implementation issues.

An alternative approach to batch estimation is the sequential estimation approach. This approach incrementally updates the estimate of the state vector as each new observation is received. Hence if $\mathbf{x}(t_0)$ is the estimate of a state vector at time t_0 based on n previous observations, then sequential estimation provides the means of obtaining a new estimate for $\boldsymbol{x}$, (i.e. $\mathbf{x}_{n+1}(t_0)$), based on $n+1$ observations by modifying the estimate $\mathbf{x}_n(t_0)$. This new estimate is obtained without revisiting all previous n observations. By contrast, in batch estimation, if a value of $\mathbf{x}_n(t_0)$ had been obtained utilizing n observations, then to determine $\mathbf{x}_{n+1}(t_0)$, all $n+1$ observations would have to be processed. The Kalman filter is a commonly used approach for sequential estimation.

Sequential estimation techniques provide a number of advantages including:

1. Determination of an estimate of the state vector with each new observation.
2. Computationally efficient scalar formulations.
3. Ability to adapt to changing observational conditions (e.g. noise, etc.).

Disadvantages of sequential estimation techniques involve potential problems in data association, divergence (in which the sequential estimator ignores new data), and problems in initiating the process. Nevertheless, sequential estimators are commonly used for tracking and positional estimation. Section 18.4 provides more detail on sequential estimation, including a process flow and discussion of implementation issues.

© 2005 by Chapman & Hall/CRC

18.3 Batch Estimation

18.3.1 Derivation of WLS Solution

In order to illustrate the formulation and processing flow for batch estimation, consider the WLS solution to a dynamic tracking problem. One or more sensors observe a moving object, reporting a total of n observations, $y_M(t_i)$, related to target position. Note: henceforth in this chapter the subscript M denotes a measured value of a quantity. The observations are assumed to be unbiased with observational noise whose standard deviation is σ_i. The (unknown) target position and velocity at time t_i is represented by an l-dimensional vector, $x(t_i)$. Since the target is moving, $\mathbf{x}$ is a function of time given by the equations of motion:

$$\mathbf{x}(t) = f(\mathbf{x}, \mathbf{x}, t) \tag{18.20}$$

Equation (18.20) represents l-simultaneous nonlinear differential equations. This is an initial value problem in differential equations. Given a set of initial conditions (viz. specification of $\mathbf{x}_0$, and $\mathbf{x}_0$ at time t_0) Equation (18.20) can be solved to predict $\mathbf{x}(t)$ at an arbitrary time t. Numerical approaches to solve (18.20) are described in a number of texts, e.g. [26,30]. Specific techniques include numerical integration methods such as Runge–Kutta methods, predictor–corrector methods, perturbation methods, or analytical solution in simple cases.

An observational model allows the observations to be predicted as a function of the unknown state vector:

$$\mathbf{y}(t) = \mathbf{g}(\mathbf{x}, t) \tag{18.21}$$

The residuals are the differences between the sensor data and the computed measurement at time t_i, given by

$$\mathbf{v}_i = [\mathbf{y}_M(t_i) - \mathbf{g}(\mathbf{x}, t_i)] \tag{18.22}$$

The WLS criteria or loss function for best fit is specified as

$$L(\boldsymbol{x}) = \sum_{i=1}^{n} \left(\frac{v_i}{\sigma_i}\right) = \sum_{i=1}^{n} \left(\frac{1}{\sigma_i^2}\right) [\mathbf{y}_M(t_i) - \mathbf{g}(\mathbf{x}, t_i)]^2 \tag{18.23}$$

$L(\boldsymbol{x})$ is a measure of the closeness of fit between the observed data and the predicted data as a function of $\mathbf{x}$. Note that if different sensor types are utilized (e.g. radar, optical tracker, etc.), then the form of the function, $\mathbf{g}(\mathbf{x}, t_i)$ changes for each sensor type. Hence, for example, $\mathbf{g}(\mathbf{x}, t_i)$ may represent one set of equations for a radar to predict range, azimuth, elevation, and range-rate, and another set of equations for an optical tracker to predict right ascension and declination angles. Further, the observation represented by $\mathbf{y}_M(t_i)$ may be a vector quantity, e.g.

$$\mathbf{y}_M(t_i) = \begin{bmatrix} \text{range}(t_i) \\ \text{range-rate}(t_i) \\ \text{azimuth}(t_i) \\ \text{elevation}(t_i) \end{bmatrix}$$

for a radar observation, or

$$\mathbf{y}_M(t_i) = \begin{bmatrix} \text{Right Ascension}(t_i) \\ \text{Declination}(t_i) \end{bmatrix}$$

for an optical tracker. In that case, $\mathbf{g}(\mathbf{x}, t_i)$ and $\boldsymbol{v}_i$ would correspondingly be vector quantities.

© 2005 by Chapman & Hall/CRC

In matrix notation, Equation (18.23) becomes

$$L(\boldsymbol{x}) = [\mathbf{y}_{\mathrm{M}} - \mathbf{g}(\mathbf{x})]^{\mathrm{T}} W [\mathbf{y}_{\mathrm{M}} - \mathbf{g}(\mathbf{x})] \tag{18.24}$$

where

$$\mathbf{y}_{\mathrm{M}} = \begin{bmatrix} y_{\mathrm{M}}(t_1) \\ y_{\mathrm{M}}(t_2) \\ \vdots \\ y_{\mathrm{M}}(t_n) \end{bmatrix}, \quad \mathbf{g}(\mathbf{x}) = \begin{bmatrix} g(x, t_1) \\ g(x, t_2) \\ \vdots \\ g(x, t_n) \end{bmatrix} \tag{18.25}$$

and

$$W = \begin{bmatrix} \dfrac{1}{\sigma_1^2} & 0 & \cdots & 0 \\ 0 & \dfrac{1}{\sigma_2^2} & \cdots & 0 \\ 0 & 0 & \cdots & \\ & & & \dfrac{1}{\sigma_n^2} \end{bmatrix} \tag{18.26}$$

The WLS estimate is the value of $\mathbf{x}_0$, denoted $\hat{\mathbf{x}}_0$, that minimizes the function $L(\mathbf{x})$.

Using an indirect approach, we seek $\hat{\boldsymbol{x}}_0$ such that

$$\frac{\partial L(\boldsymbol{x})}{\partial \mathbf{x}} = 2[\mathbf{y}_{\mathrm{M}} - \mathbf{g}(\mathbf{x})]^{\mathrm{T}} W \frac{\partial \mathbf{g}}{\partial \mathbf{x}} = 0 \tag{18.27}$$

Equation (18.27) represents a set of nonlinear equations in l unknowns (the l components of $\mathbf{x}_0$).

The solution to Equation (18.27) may be obtained by a multi-dimensional Newton–Raphson approach as indicated in the previous section. The following derivation explicitly shows the linearized, iterative solution. First, expand the measurement prediction, $\mathbf{g}(\mathbf{x}, t)$, in a Taylor series about a reference solution $\mathbf{x}_{\mathrm{REF}}(t)$, namely

$$\mathbf{g}(\mathbf{x}, t_i) = \mathbf{g}(\mathbf{x}_{\mathrm{REF}}, t) + H_i \Delta \mathbf{x}_i \tag{18.28}$$

where

$$\Delta \mathbf{x}_i = \boldsymbol{x}(t_i) - \mathbf{x}_{\mathrm{REF}}(t_i) \tag{18.29}$$

and

$$H_i = \frac{\partial \mathbf{g}(x, t_i)}{\partial \mathbf{x}} \tag{18.30}$$

The value $\mathbf{x}_{\mathrm{REF}}$ is a particular solution to the dynamical equation of motion [Equation (18.20)].

© 2005 by Chapman & Hall/CRC

A further simplification may be obtained if the equation of motion is linearized, i.e.

$$\Delta \mathbf{x}_i = \Phi(t_i, t_j)\Delta \mathbf{x}_j \tag{18.31}$$

where $\Phi(t_i, t_j)$ represents a state transition matrix which relates variations about $\mathbf{x}_{\text{REF}}(t)$ at times t_i, and t_j:

$$\Phi(t_i, t_j) = \frac{\partial \mathbf{x}(t_i)}{\partial \mathbf{x}(t_j)} \tag{18.32}$$

Substituting Equation (18.32) into Equation (18.28) yields

$$\mathbf{g}(\mathbf{x}, t_i) = \mathbf{g}(\mathbf{x}_{\text{REF}}, t_i) + H_i \Phi(t_i, t_j)\Delta \mathbf{x}_j \tag{18.33}$$

Using this expression for $\mathbf{g}(\mathbf{x}, t_i)$ in Equation (18.27) yields

$$[\Delta \mathbf{y} - H(\mathbf{x}_j)\Delta x_j]^{\mathrm{T}} W H(\mathbf{x}_j) = 0 \tag{18.34}$$

where

$$\Delta \mathbf{y} = \mathbf{y}_{\text{M}} - \mathbf{g}(\mathbf{x}_{\text{REF}}) = \begin{bmatrix} y_{\text{M}}(t_1) - g(x_{\text{REF}}, t_1) \\ y_{\text{M}}(t_2) - g(x_{\text{REF}}, t_2) \\ \vdots \\ y_{\text{M}}(t_n) - g(x_{\text{REF}}, t_n) \end{bmatrix} \tag{18.35}$$

and

$$H(\mathbf{x}_j) = \begin{bmatrix} H_1 \Phi(t_1, t_j) \\ H_2 \Phi(t_2, t_j) \\ \vdots \\ H_n \Phi(t_n, t_j) \end{bmatrix} \tag{18.36}$$

Solving equation (18.34) for $\Delta \mathbf{x}_j$ yields

$$\Delta \hat{\mathbf{x}}_j = [H(\mathbf{x}_j)^{\mathrm{T}} W H(\mathbf{x}_j)]^{-1} [H(\mathbf{x}_j)^{\mathrm{T}} W \Delta \mathbf{y}] \tag{18.37}$$

The increment $\Delta \mathbf{x}_j$ is added to $\mathbf{x}_{\text{REF}}(t_j)$:

$$\hat{\mathbf{x}}(t_j) = \mathbf{x}_{\text{REF}}(t_i) + \Delta \hat{\mathbf{x}}_j \tag{18.38}$$

to produce an updated estimate of the state vector. The improved value of $\mathbf{x}(t_j)$ is used as a new reference value of $\mathbf{x}(t_j)$. Equations (18.37) and (18.38) are applied iteratively until $\Delta \mathbf{x}_j$ becomes arbitrarily small.

© 2005 by Chapman & Hall/CRC

Table 18.4. Summary of batch estimation solutions

Optimization criterion	Mathematical formulation	Linearized iterative solution
Least squares	$L(\boldsymbol{x}) = \mathbf{v}\mathbf{v}^{\mathrm{T}}$	$\Delta\hat{\mathbf{x}}_j = [H^{\mathrm{T}}H]^{-1}[H^{\mathrm{T}}\mathbf{v}]$
Weighted least squares	$L(\boldsymbol{x}) = \mathbf{v}w\mathbf{v}^{\mathrm{T}}$	$\Delta\hat{\mathbf{x}}_j = [H^{\mathrm{T}}WH]^{-1}[H^{\mathrm{T}}W\mathbf{v}]$
Bayesian weighted least squares	$L(\boldsymbol{x}) = vwv + (\mathbf{x}-\mathbf{x}_0)P_{\Delta\mathbf{x}_0}(\mathbf{x}-\mathbf{x}_0)^{\mathrm{T}}$	$\Delta\hat{\mathbf{x}}_j = [H^{\mathrm{T}}WH + P_{\Delta x}^{-1}][H^{\mathrm{T}}W\mathbf{v} + P_{\Delta x}^{-1}\Delta\hat{\mathbf{x}}_{j-1}]$
Maximum likelihood estimate	$L(x) = \prod_{i=1}^{n} l_i(n_i/x)$	$\Delta\hat{\mathbf{x}}_j = [H^{\mathrm{T}}M^{-1}H]^{\mathrm{T}}[H^{\mathrm{T}}M^{-1}\mathbf{v}]$

Table 18.4 illustrates the linearized iterative solution for several optimization criteria, including LS, WLS, BWLS and the maximum likelihood optimization criteria. In Equation (18.37), $\Delta\mathbf{y}$ represents the difference between predicted and actual observations, while $H(\mathbf{x}_j)$ expresses the relationship between changes in predicted observations and changes in components of the state vector. For static (nondynamic) problems, the state vector $\boldsymbol{x}(t)$ is constant in time, and the state transition matrix reduces to the identity matrix:

$$\Phi(t_i, t_j) = I = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ & & \ddots & \\ 0 & 0 & \cdots & 1 \end{bmatrix} \tag{18.39}$$

18.3.2 Processing Flow

The processing flow to solve the batch estimation problem is illustrated in Figure 18.4 for a WLS formulation. The indirect approach discussed in the previous section is used. Inputs to the process include an initial estimate of the state vector $\hat{\boldsymbol{x}}_0$ at an epoch time t_0, and n observations $\mathbf{y}_{\mathrm{M}k}$ at times t_k, with associated uncertainties $\boldsymbol{\sigma}_k$. The output of the process is an improved estimate of the state vector $\boldsymbol{x}_{i+1}(t_0)$ at the epoch time t_0.

The processing flow shown in Figure 18.4 utilizes two nested iterations. An inner iteration (or processing loop letting $k = 1, 2, \ldots, n$), cycles through each observation $y_{\mathrm{M}k}$ performing a series of calculations:

1. Retrieve the kth observation $\mathbf{y}_{\mathrm{M}k}$, its associated time of observation t_k, and observational uncertainty σ_k.
2. Solve the differential equation of motion [Equation (18.20)] to obtain $\boldsymbol{x}_{ki}(t_k)$. That is, using the current estimate of the state vector, $\boldsymbol{x}_i(t_0)$, update to time t_k.
3. Compute a predicted observation, $\mathbf{g}[\mathbf{x}_i(t_k), t_k]$ based on sensor models. Note that the predicted observation utilizes a model appropriate to the sensor type (e.g. radar, optical, etc.).
4. Compute the transition matrix $\Phi(t_k, t_0)$.
5. Calculate the quantity $H_k\Phi(t_k, t_0)$, and the observation residual $\delta\mathbf{y}_k = [\,\mathbf{y}_{\mathrm{M}}(t_k) - \mathbf{g}(\mathbf{x}_i, t_k)]$.
6. Accumulate the matrices.

$$A = A + \frac{H_k^{\mathrm{T}}H_k}{\sigma_k^2} \tag{18.40}$$

and

$$B = B + \frac{H_k^{\mathrm{T}}\delta\mathbf{y}_k}{\sigma_k^2} \tag{18.41}$$

Steps (1) through (6) are performed for each observation $k = 1, 2, \ldots, n$.

© 2005 by Chapman & Hall/CRC

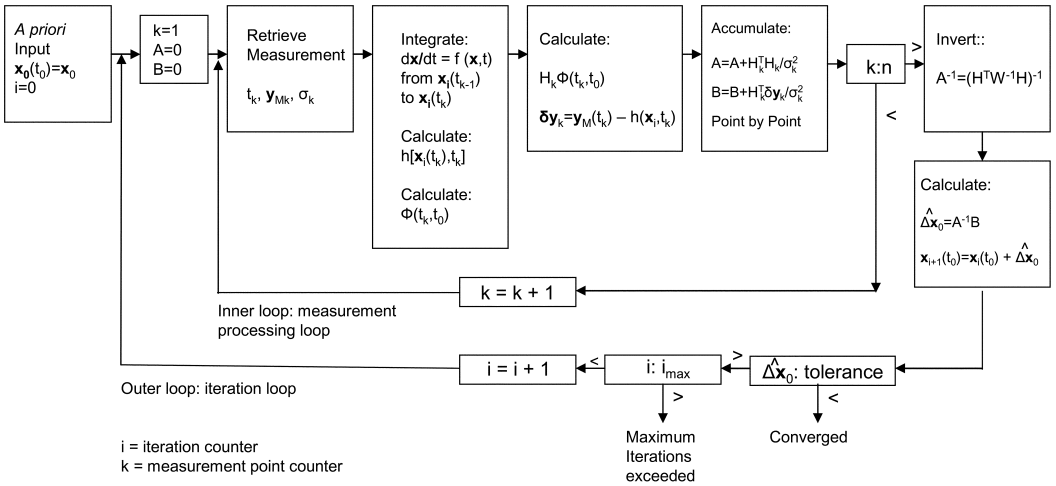


Figure 18.4. Weighted least squares batch process computational flow sequence.

An outer processing loop ($i = 1, 2, \ldots$) iteratively computes and applies corrections to the state vector until convergence is achieved. For each iteration, we compute

$$A^{-1} = (H^{\mathrm{T}} W^{-1} H)^{-1} \tag{18.42}$$

and

$$\Delta\hat{\mathbf{x}}_0 = A^{-1} B \tag{18.43}$$

with

$$\mathbf{x}_{i+1}(t_0) = \mathbf{x}_i(t_0) + \Delta\hat{\mathbf{x}}_0 \tag{18.44}$$

Hence, the state vector is successively improved until $\Delta\hat{\mathbf{x}}_0$ becomes arbitrarily small.

The process described here, and illustrated in Figure 18.4 is meant to convey the essence of the batch solution via a linearized iterative approach. It can be seen that the solution may become computationally demanding. For each observation (which may number in the hundreds to thousands), we must solve a nonlinear set of differential equations, perform coordinate transformations to predict an observation, compute the transition matrix, and perform several matrix multiplications. These calculations are performed for all n observations. Moreover, the complete set (for all n observations) of computations is iteratively performed to achieve an estimate of $\mathbf{x}(t_0)$. Upwards of 10 to 30 iterations of $\Delta\mathbf{x}_j$ may be required to achieve convergence, depending upon the initial value chosen for $\mathbf{x}(t_0)$.

18.3.3 Batch Processing Implementation Issues

The processing flow for batch estimation shown in Figure 18.4 is meant to be illustrative rather than a prescription or flowchart suitable for software implementation. Several implementation issues often arise. We will discuss a few of these issues, including convergence, data editing, the initial estimate of $\mathbf{x}$, and observability.

© 2005 by Chapman & Hall/CRC

The processing flow in Figure 18.4 shows an outer processing loop in which successive improvements are made to the state vector estimate. The convergence criteria tests the magnitude of $\Delta\mathbf{x}$, and declares convergence when

$$|\Delta\mathbf{x}_0| \leq \boldsymbol{\varepsilon} \tag{18.45}$$

The iterations are terminated when the incremental changes to the state vector fall within an arbitrarily small increment (for each component of the state vector). This is a logical criterion, since we can use physical arguments to establish the values for ε. For example, we might declare that distances within 1 m, velocities within 1 cm/s, frequencies within 1 Hz, etc. are arbitrarily small. Other convergence criteria might equally well be used. An example is the ratio criterion

$$\left|\frac{\Delta\mathbf{x}_0}{\mathbf{x}_0}\right| \leq \boldsymbol{\varepsilon} \tag{18.46}$$

A number of convergence criteria have been used including multiple logical conditions [e.g. Equation (18.45) or (18.46), etc.). There is no guarantee that the iterative solution will converge in a finite number of iterations. Thus, logical checks should be made to determine how many iterations have been performed with some upper bound (e.g. $k \leq 50$) established to terminate the iterations.

In batch estimation it is tempting to perform data editing within the outer processing loop. A common practice is to reject all observations for which the residual ($|\boldsymbol{v}_k|$) exceeds either an *a priori* limit or a standard deviation (e.g. 3σ) test. Such a practice is fraught with potential pitfalls. Deutsch[19] discusses some of these pitfalls. Two problems are most notable. Iterative data editing can prolong the outer-loop iteration. It is possible in the ith estimate of $\mathbf{x}(t_0)$ to reject one or more observations only to find in the $(i+1)$st estimate that these observations are acceptable. Hence, the iteration for $\mathbf{x}(t_0)$ can sometimes oscillate, alternatively rejecting and accepting observations. A second problem involves the case in which all observations are valid and highly accurate. Statistically, there will be some valid observations whose residuals exceed 3σ. Rejecting or editing out such observations corrupts the solution for $\mathbf{x}(t_0)$ by rejecting perfectly good data. A rule of thumb is not to reject any data unless these are valid physical reasons for such editing.

Another implementation issue for batch processing involves how to obtain an initial estimate of the state vector $\mathbf{x}_0(t_0)$. Generally, several observations may be used in a minimum data solution to obtain a value of $\mathbf{x}(t_0)$. Alternatively, some sensors may provide estimates of the state vector, or an estimate of $\mathbf{x}$ may be available from other *a priori* information. The generation of a starting value for $\mathbf{x}$ is very much dependent upon the particular sensors and observing geometry. Sometimes, several minimum data sets are used (i.e. observations $\mathbf{y}_{\mathrm{M}}$, $\mathbf{y}_{\mathrm{M3}}$, and $\mathbf{y}_{\mathrm{M5}}$; observations $\mathbf{y}_{\mathrm{M2}}$, $\mathbf{y}_{\mathrm{M4}}$, and $\mathbf{y}_{\mathrm{M6}}$) with an initial estimate of $\mathbf{x}(t_p)$ developed from each data set. Subsequently, the initial estimates are averaged to produce a starting value for the estimation process.

A final implementation issue is the question of observability. We briefly introduced this issue in Section 18.4. Observability is the question of whether improvements in components of $\mathbf{x}(t)$ can be obtained based on the observational data. Mathematically, this problem is exhibited via an ill-conditioned matrix:

$$[H^{\mathrm{T}}W^{-1}H] \tag{18.47}$$

Several methods have been introduced to address such an ill-conditioned system. For example, we may require that the determinant of matrix $H^{\mathrm{T}}H$ be nonzero at each iterative step. Alternatively, nonlinear terms may be introduced to treat the ill-conditioned linear system. While these techniques may be useful, there is no substitute for care in selecting and analyzing the choice of state vector components.

© 2005 by Chapman & Hall/CRC

18.4 Sequential Estimation and Kalman Filtering

During the early 1960s, a number of technical papers were published describing a sequential approach to estimation [31–33]. These papers described a linearized sequential technique to update an estimate of a state vector. The work was a discrete implementation of earlier continuous formulations by Wiener [8] and Kolmogorov [9] in the late 1940s. An informal history of the development of the Kalman filter, and its application to space flight, is provided by McGee and Schmidt [34]. Since then, much work has been performed on discrete estimation. A number of techniques exist. An extensive review of recursive filtering techniques is provided by Sayed and Kailath [35], who compare variants of the Kalman filter, including the covariance Kalman filter, the information filter, the square-root covariance filter, the extended square-root information filter, the square-root Chandrasekhar filter, and the explicit Chandrasekhar filter. We will describe one such approach utilizing the Kalman filter.

Several approaches may be used to derive the sequential estimation equations. In particular, the reader is referred to the feedback-control system approach described by Gelb [12]. In this section, we will derive the equations for a WLS optimization criterion, for a dynamic tracking problem, identical to that in the previous section. Following a derivation, we will present a processing flow and discussion of implementation issues.

18.4.1 Deviation of Sequential WLS Solution

Assume that n observations are available from multiple sensors, and that a WLS solution has been obtained, i.e.

$$\Delta\hat{\mathbf{x}}_n = [H_n W_n^{-1} H_n]^{-1} H_n^{\mathrm{T}} W_n^{-1} \Delta\mathbf{y}_n \tag{18.48}$$

H_n denotes $H(\mathbf{x}_n)$ the partial derivatives of the observation components with respect to the state vector [Equation (18.36)], and $\Delta\mathbf{y}_n$ denotes the difference between the measured and predicted observation [Equation (18.22)]. Suppose that one additional observation $\mathbf{y}_{\mathrm{M}}(t_{n+1})$ is received. How does the $(n+1)$st observation affect the estimate of $\mathbf{x}$? Let us utilize the WLS formulation separating the $(n+1)$st data from the previous n observations. Thus

$$\Delta\hat{\boldsymbol{x}}_{n+1} = [H_{n+1} W_{n+1}^{-1} H_{n+1}]^{-1} H_{n+1}^{\mathrm{T}} W_{n+1}^{-1} \Delta y_{n+1} \tag{18.49}$$

where

$$H_{n+1} = \left|\frac{H_n \phi(t_n, t_{n+1})}{H_{n+1}}\right|; \quad W_{n+1} = \begin{pmatrix} W_n & 0 \\ 0 & \sigma_{n+1}^2 \end{pmatrix}; \quad \Delta\mathbf{y}_{n+1} = \begin{vmatrix} \Delta y_n \\ \delta y_{n+1} \end{vmatrix} \tag{18.50}$$

and the $(n+1)$st residual is

$$\delta\mathbf{y}_{n+1} = \mathbf{y}_{\mathrm{M}}(t_{n+1}) - \mathbf{g}(\mathbf{x}_{\mathrm{REF}}, t_{n+1}) \tag{18.51}$$

Substituting Equations (18.50) into Equation (18.49) yields

$$\Delta\hat{\mathbf{x}}_{n+1} = \left[[H_n\phi,\ H_{n+1}]\begin{pmatrix} w_n^{-1} & 0 \\ 0 & \sigma_{n+1}^{-2} \end{pmatrix}\begin{bmatrix} H_n\phi \\ H_{n+1} \end{bmatrix}\right]^{-1} [H_n\phi,\ H_{n+1}]\begin{pmatrix} w_n^{-1} & 0 \\ 0 & \sigma_{n+1}^{-2} \end{pmatrix}\begin{bmatrix} \Delta y_m \\ \delta y_{n+1} \end{bmatrix} \tag{18.52}$$

which can be manipulated to obtain

$$\Delta\hat{\mathbf{x}}(t_{n+1}/t_n) = \Delta\hat{\boldsymbol{x}}(t_{n+1}/t_n) - K[H_{n+1}\Delta\hat{\mathbf{x}}(t_{n+1}/t_n) - \delta\mathbf{y}_{n+1}] \tag{18.53}$$

© 2005 by Chapman & Hall/CRC

where

$$K = P_n(t_{n+1})H_{n+1}^{\mathrm{T}}\left[H_{n+1}P_n(t_{n+1})H_{n+1}^{\mathrm{T}} + \boldsymbol{\sigma}_{n+1}^2\right]^{-1} \tag{18.54}$$

$$P_{n+1}(t_{n+1}) = P_n(t_{n+1}) - KH_{n+1}P_n(t_{n+1}) \tag{18.55}$$

$$\Delta\hat{\mathbf{x}}(t_{n+1}/t_n) = \Phi(t_{n+1}, t_n)\Delta\hat{\mathbf{x}}(t_n/t_n) \tag{18.56}$$

$$P_n(t_{n+1}) = \Phi(t_{n+1}, t_n)P_n(t_n)\Phi \tag{18.57}$$

Equations (18.53) to (18.57) constitute a set of equations for recursive update of a state vector. Thus, given n observations $\mathbf{y}_{\mathrm{M}}(t_n)$ $(t = 1, 2, \ldots, n)$, and an associated estimate for $\mathbf{x}(t)$, these equations prescribe an update of $y_{\mathrm{M}}(t_n)$ based on a new observation $\mathbf{y}_{\mathrm{M}}(t_{n+1})$. Clearly the equations can be applied recursively, replacing the solution for $\mathbf{x}_n(t_n)$ by $\mathbf{x}_{n+1}(t_{n+1})$ and processing yet another observation $y_{\mathrm{M}}(t_{n+2})$, etc.

Equation (18.53) is a linear expression that expresses the updated value of $\mathbf{x}(t_{n+1})$ as a function of the previous value $\mathbf{x}(t_n)$, a constant K, and the observation residual $\delta\mathbf{y}_{n+1}$. The constant K is called the Kalman gain, which in turn is a function of the uncertainty in the state vector (viz. the covariance of x_n given by $P_n(t_{n+1})$, and the uncertainty in the observation $\boldsymbol{\sigma}_{n+1}$). Equation (18.55) updates the uncertainty in the state vector, while Equation (18.56) uses the transition matrix to update the value of x from time, t_n to time t_{n+1}. Similarly, Equation (18.57) updates the covariance matrix at time t_n to time t_{n+1}. In these equations, the parenthetical expression (t_{n+1}/t_n) denotes that the associated quantity is based on the previous n observations, but is extrapolated to time t_{n+1}. Correspondingly, (t_{n+1}/t_{n+1}) indicates that the associated quantity is valid at time t_{n+1} and also has been updated to include the effect of the $(n+1)$st observation.

The Kalman gain K directly scales the magnitude of the correction to the state vector $\Delta\mathbf{x}_n$. The gain K will be relatively large (and hence will cause a large change in the state vector estimate) under two conditions:

1. When the uncertainty in the state vector is large (i.e. when $P_n(t_{n+1})$ is large).
2. When the uncertainty in the $(n+1)$st observation is small (i.e. when σ_{n+1} is small).

Conversely, the Kalman gain will be small when the state vector is well known (i.e. when $P_n(t_{n+1})$ is small) and/or when the $(n+1)$st observation is very uncertain (i.e. when $\boldsymbol{\sigma}_{n+1}$ is large). This result is conceptually pleasing, since we want to significantly improve inaccurate state vector estimates with accurate new observations but do not want to corrupt accurate state vectors with inaccurate data.

There are two main advantages of sequential estimation over batch processing. First, Equations (18.53) to (18.57) can be formulated entirely as scalar equations requiring no matrix inversions. Even when each observation $\mathbf{y}_{\mathrm{M}}(t_n)$ is a vector quantity (e.g. a radar observation comprising range, range-rate, azimuth, and elevation), we can treat each observation component as a separate observation occurring at the same observation time t_{n+i}. This scalar formulation allows very computationally efficient formulations. The second advantage is that the sequential process allows the option of updating the reference solution $(\mathbf{x}_{\mathrm{REF}}(t_0))$ after each observation is processed. This option is sometimes referred to as the extended Kalman filter. This option provides an operational advantage for dynamic problems, such as target tracking, because a current estimate of $\mathbf{x}(t)$ is available for targeting purposes or to guide the sensors.

18.4.2 Sequential Estimation Processing Flow

A processing flow for sequential estimation is shown in Figure 18.5. The processing flow is shown for an extended Kalman filter with dynamic noise. Inputs to the process are an initial estimate of the state

© 2005 by Chapman & Hall/CRC

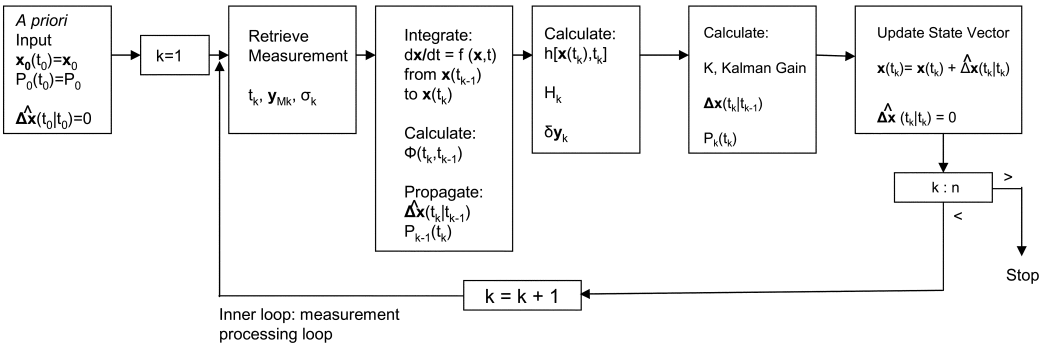


Figure 18.5. Recursive filter process with dynamic noise (extended Kalman filter).

vector $\mathbf{x}_0(t_0)$ and associated covariance matrix $P_0(t_0)$. For each measurement $k = 1, 2, \ldots, n$, the following steps are performed:

1. Retrieve the observation $\mathbf{y}_M(t_k)$ and its associated uncertainity σ_k.
2. Solve the differential equations of motion [Equation (18.20)] to propagate the state vector from time t_{k-1} to t_k.
3. Compute the transition matrix $\Phi(t_k/t_{k-1})$.
4. Propagate $\Delta\mathbf{x}$ and P_{k-1} from time t_{k-1} to t_k using the transition matrix [i.e. Equations (18.56) and (18.57) respectively].
5. Compute a predicted observation $\mathbf{g}(\mathbf{x}(t_k), t_k)$ [Equation (18.21)], the observation residual δ_{vk}, and H_k, the partial derivative of the predicted observation with respect to the state vector.
6. Compute the Kalman gain via Equation (18.54).
7. Update the state vector correction $\Delta\mathbf{x}(t_k, t_k)$ [Equation (18.56)] and the covariance matrix $P_k(t_k)$ [Equation (18.55)].
8. Update the reference state vector

$$\hat{\mathbf{x}}(t_k) = \hat{\mathbf{x}}(t_k) + \Delta\hat{\mathbf{x}}(t_k/t_k) \tag{18.58}$$

Steps (1) through to (8) are repeated until all observations have been processed.

Output from the sequential estimation is an updated state vector $\mathbf{x}(t_k)$ and associated covariance matrix $P_n(t_n)$ based on all n observations.

The concept of how the Kalman filter updates the state vector and associated quantities from one observation to the next is illustrated in Figure 18.6 (adapted from Gelb [12]).

Figure 18.6 illustrated a timeline with two observations at time t_{k-1} and t_k. At time t_{k-1} we have an estimate of the state vector $\mathbf{x}_{k-1}(-)$ and its associated covariance $P_{k-1}(-)$. The parenthetical use of the minus sign indicates that the values of the state vector and covariance matrix are the estimates at time t_{k-1}, but prior to incorporating the effect of the new observation. When the observation at time t_k is received, the Kalman equations are used to update to estimate a new value of the state vector. The values of the Kalman gain K and the observation uncertainty $\mathbf{R}$ are used to develop updates to the state vector $\boldsymbol{x}$ and to the covariance matrix. At a subsequent time t_k we receive a new observation. It is first necessary to propagate the state vector $\boldsymbol{x}_{k-1}$, and the covariance matrix P_{k-1} forward in time from time t_{k-1} to time t_k. Using the equations of motion for the state vector and the covariance matrix does this. As a result, we have an estimate of the state vector $\boldsymbol{x}$ at time t_k based on the $k-1$ processed observations. At time t_k we again apply the Kalman update equations to result in improved estimates of the state vector and covariance matrix due to the new information (via the observation at time t_k). This process continues until all observations are processed.

© 2005 by Chapman & Hall/CRC

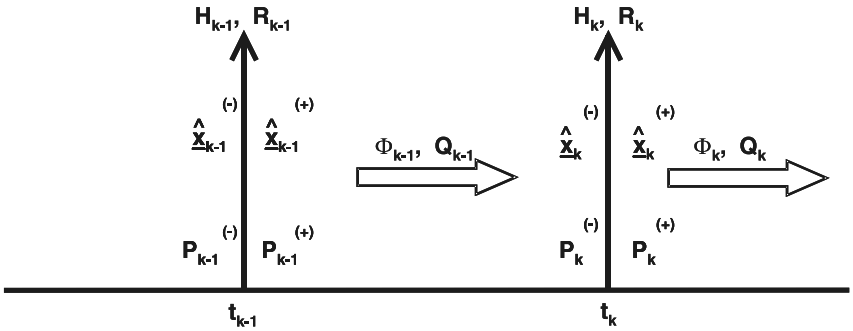


Figure 18.6. Kalman filter update process [12]. The Kalman filter equations are used as each new observation is received to update the estimates of the state vector and associated covariance matrix.

18.5 Sequential Processing Implementation Issues

While implementation of a sequential estimation process can be performed in a computationally efficient, scalar formulation, there are a number of other implementation issues. Some of these issues are described below.

18.5.1 Filter Divergence and Process Noise

One potential problem of a sequential estimation is termed divergence. The problem occurs when the magnitude of the state vector covariance matrix becomes relatively small. This decrease in $P_{\Delta x}$ occurs naturally as more observations are processed, since the knowledge of the state vector increases with the number of observations processed. When $P_{\Delta x}$ becomes relatively small, the Kalman gain becomes correspondingly small [since $P_{\Delta x}$ is a multiplicative factor in Equation (18.43)]. The result is that the estimator *ignores* new data and does not make significant improvements to $\Delta\mathbf{x}$. While this would seem to be a desirable result of the estimation process, sometimes $P_{\Delta x}$ becomes artificially small, resulting in the filter disregarding valid observations. In order to correct this divergence problem, two techniques are often used: (1) introduction of process noise and (2) use of a fading memory factor.

Process or dynamic noise is white noise added to the linear state perturbation model, i.e.

$$\Delta\hat{\mathbf{x}}(t_{n+1}) = \Phi(t_{n+1}, t_n)\Delta\hat{\mathbf{x}}(t_n) + \boldsymbol{\eta} \tag{18.59}$$

The vector $\boldsymbol{\eta}$ is assumed to have zero mean and covariance Q

$$\text{cov}(\boldsymbol{\eta}) = Q \tag{18.60}$$

This dynamic noise represents imperfections or random errors in the dynamic state model. The noise vector in turn affects the propagation of the covariance propagation equation, yielding

$$P_n(t_{n+1}) = \Phi(t_{n+1}, t_n)P_n(t_n)\Phi(t_{n+1}, t_n)^{\mathrm{T}} + Q(t_n) \tag{18.61}$$

instead of Equation (18.57). Hence, the covariance matrix has a minimum magnitude of $Q(t_n)$. This reduces the divergence problem.

Another technique used to avoid divergence is the use of a fading memory. The concept here is to weight recent data more than older data. This can be accomplished by multiplying the covariance matrix by a memory factor:

$$s = \mathrm{e}^{\Delta t/\tau}$$

© 2005 by Chapman & Hall/CRC

such that

$$P_n(t_{n+1}) = s\Phi(t_{n+1}, t_n)P_n(t_n)\Phi(t_{n+1}, t_n)^{\mathrm{T}} \tag{18.62}$$

where Δt is the interval $(t_{n+1}, -t_n)$, and τ is an *a priori* specified memory constant. This is chosen so that $s \geq 1$. The fading memory factor s amplifies or scales the entire covariance matrix. By contrast, process noise is additive and establishes a minimum value for P_n. Either of these techniques can effectively combat the divergence problem. Their use should be based on physical or statistical insight into the estimation problem.

18.5.2 Filter Formulation

A second implementation issue for sequential estimation involves the formulation of the filter equations. The previous section derived a linearized set of equations. Nonlinear formulations can also be developed. Gelb [12] describes a second-order approximation to sequential estimation. Even if a linear approximation is used, there are variations possible in the formulation of the equations. A number of techniques have been developed to ensure numerical stability of the estimation process. Gelb [12] describes a so-called square root formulation and a UDU^{T} formulation both aimed at increasing the stability of the estimators. It is beyond the scope of this chapter to describe these formulations in detail. Nevertheless, the reader is referred to work by Tapley and Peters [36], for example, for a detailed discussion and comparison of results.

The issue of observability is just as relevant to sequential estimation as it is to batch estimation. The state vector may be only weakly related to the observational data, or there may exist a high degree of correlation between the components of the state vector. In either case, the state vector will be indeterminate based on the observational data. As a result, the filter will fail to obtain an accurate estimate of the state parameters.

A final implementation addressed here is that of data editing. As in the batch estimation process, indiscriminate editing (rejection) of data is not recommended without a sound physical basis. One technique for editing residuals is to compare the magnitude of the observation residual $\mathbf{v}(t_k)$ with the value $[HPH^{\mathrm{T}} + \sigma_{\mathrm{OBS}}]$

$$\frac{|\boldsymbol{v}(t_k)|}{[HPH^{\mathrm{T}} + \boldsymbol{\sigma}^2_{\mathrm{OBS}}]} > O_{\mathrm{MAX}} \tag{18.63}$$

If the ratio specified by Equation (18.63) exceeds O_{MAX}, then the observation is rejected and the state vector and covariance matrix are not updated.

18.5.3 Maneuvering Targets

One of the special difficulties with estimation techniques for target tracking involves maneuvering targets. The basic problem is that acceleration cannot be observed directly. Instead, we can only observe the after-the-fact results of an acceleration maneuver (e.g. turn, increase in speed). So, in one sense, the estimate of a target's state vector "lags behind" what the target is actually doing. For benign maneuvers, this is not a large problem, since, given new observations, the estimated state vector "catches up" with the actual state of the observed target. A challenge occurs, however, if one or more targets are deliberately maneuvering to avoid detection or accurate tracking (e.g. terrain-following tactical aircraft or aircraft involved in a "dog fight"). Because a sequential estimator such as a Kalman filter processes data on an observation-by-observation basis, it is possible for a sequential estimator to be very challenged by such maneuvers. Examples of work in this area include the studies of Schultz *et al.* [37], McIntyre and Hintz [38] and Lee and Tahk [39]. A series of surveys of maneuvering target methods has been conducted by Jilkov [40–43].

© 2005 by Chapman & Hall/CRC

Table 18.5. Summary of Kalman filter software tools

Tool name	Vendor	Website reference
KFTool 2.5.1	Navtech	http://www.navtechgps.com/supply/kftool.asp
M-KFTOOL	MATLAB	http://www.navtechgps.com/pdf/mkftoolcontents.pdf
ReBEL	Machine Learning & Signal Processing Group	http://choosh.ece.ogi.edu/rebel/index.html
MathLibX	Newcastle Scientific	http://www.virtualsoftware.com/ProdPage.cfm?ProdID=1405
Bayesian Filtering	Australian Center for Field Robotics	http://www.acfr.usyd.edu.au/technology/bayesianfilters/Bayes++.htm
Flight Dynamics S/S	Telesat	http://www.telsat.ca/eng/international_software1.htm
IMSL	Visual Numerics Inc.	http://www.vni.com/press/cn150.html

Several methods have been used to try to address such maneuvering targets. These include:

1. *Estimate the acceleration.* One way to address the problem is to augment the state vector to include acceleration terms (e.g. add components of acceleration to the state vector $\boldsymbol{x}$). This poses a problem in the estimation process because components of acceleration are highly correlated with components of velocity. Hence, the filter can become numerically unstable.
2. *Model the acceleration and estimate the model parameters.* Various researchers have attempted to develop models for acceleration using assumptions such as the target only performs horizontal maneuvers at fixed turn rates, etc. Under these assumptions, the state vector can be augmented to include parameters in the acceleration model.
3. *Detect maneuvers and use decision-based methods to select an acceleration model.* Another technique used to address maneuvers is to monitor the size of the observation residuals to try to detect when a maneuver has occurred (based on anomalies in the predicted versus actual observations). Subsequently, automated decision methods can be used to select the appropriate acceleration model based on the observed observation residual anomalies.
4. *Use multiple Kalman filters in parallel (each having a different model of acceleration).* Finally, some researchers have used an approach in which multiple estimators are used in parallel, each using a different model of acceleration. In this case, each estimator provides a different estimate (as a function of time) of the target's position. An expert system ''overseer'' can be used to monitor the observation residuals to determine which model is ''correct'' at a given time.

Fundamentally this remains a challenge for any estimation process because the sought for state (viz. knowledge of the acceleration) cannot be directly observed.

18.5.4 Software Tools

There are a wide variety of numerical methods libraries, mathematical software, simulation tools, and special toolkits that support the development of Kalman filters and related estimation algorithms. A sample of software tools is provided in Table 18.5. A survey of various estimation software tools is available at the Website: http://www.lionhrtpub.com/orms/surveys/FSS/fss9.html.

Acknowledgments

All figures and tables in this chapter are taken from [1]. Reprinted with permission of Artech House.

References

[1] Hall, D., *Mathematical Techniques in Multisensor Data Fusion*, Artech House Inc., Norwood, MA, 1992.

© 2005 by Chapman & Hall/CRC

[2] Hall, D. and McMullen, S., *Mathematical Techniques in Multisensor Data Fusion*, 2nd ed. Artech House Inc., Boston, MA, 2004.
[3] Sorenson, H.W., Least squares estimation: from Gauss to Kalman, *IEEE Spectrum*, 63, 1970.
[4] Gauss, K.G., *Theory of Motion of the Heavenly Bodies*, Dover, New York, 1963.
[5] Legendre, L.M., *Nouvelles Methods pour la Determination des Orbits des Commetes*, Paris, 1806.
[6] Bell, E.T., *Men of Mathematics*, Simon and Schuster, New York, 1961.
[7] Fischer, R.A., On the absolute criteria for fitting frequency curves, *Messenger of Mathematics*, 41, 155, 1912.
[8] Wiener, N., *The Extrapolation, Interpolation and Smoothing of Stationary Time Series*, John Wiley and Sons, New York, 1949.
[9] Kolmogorov, A.N., Interpolation and extrapolation von stationaren zufalliegen Folgen, *Bulletin of the Academy of Sciences, USSR, Ser. Math. S.*, 3–14, 1941.
[10] Kalman, R.E., New methods in Wiener filtering theory, in *Proceedings of the First Symposium of Engineering Application of Random Function Theory and Probability*, John Wiley and Sons, New York, 270, 1963.
[11] Blackman, S.S., *Multiple Target Tracking with Radar Applications*, Artech House Inc., Norwood, MA, 1986.
[12] Gelb, A., *Applied Optimal Estimation*, MIT Press, Cambridge, MA, 1974.
[13] Ramachandra, K.V., *Kalman Filtering Techniques for Radar Tracking*, Marcel Dekker, New York, 2000.
[14] Grewal, M.S. and Andrews, A.P., *Kalman Filtering Theory and Practice using MATLAB*, John Wiley and Sons, New York, 2001.
[15] Kessler, O. *et al.*, Functional description of the data fusion process, Office of Naval Technology, Naval Air Development Center, Warminster, PA, 1992.
[16] Steinberg, A., Revisions to the JDL data fusion process model, in *Handbook of Multisensor Data Fusion*, Hall, D. and Llinas, J., (eds), Boca Raton, FL, CRC Press, 2-1, 2001.
[17] Zavaleta, E.L. and Smith, E.J., Goddard Trajectory Determination System User's Guide, Computer Sciences Corporation, Silver Spring, MD, 1975.
[18] Hall, D. and Waligora, S.R., Orbit/attitude estimation using Landsat-1 and Landsat-2 landmark data, in *NASA Goddard Space Flight Center Flight Mechanics/Estimation Theory Symposium*, NASA Goddard Space Flight Center, MD, NASA, 1978.
[19] Deutsch, R., *Orbital Dynamics of Space Vehicles*, Prentice-Hall, Englewood Cliffs, NJ, 1963.
[20] Waltz, E. and Llinas, J., *Multi-Sensor Data Fusion*, Artech House Inc., Norwood, MA, 1990.
[21] Escobol, R.R., *Methods of Orbit Determination*, Krieger, Melbourne, FL, 1976.
[22] Kamen, E.W. and Sastry, C.R., Multiple target tracking using an extended Kalman filter, in *SPIE Signal and Data Processing of Small Targets*, SPIE, Orlando, FL, 1990.
[23] Sanza, N.D. *et al.*, Spherical target state estimators, *American Control Conference*, Baltimore, MD, 1994.
[24] Wilde, D.J. and Beightler, C.S., *Foundations of Optimization*, Prentice-Hall, Englewood Cliffs, NJ, 1967.
[25] Press, W.H. *et al.*, *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press, New York, 1986.
[26] Brent, R.P., *Algorithms for Minimization without Derivatives*, Prentice-Hall, Englewood Cliffs, NJ, 1973.
[27] Shoup, T.E., *Optimization Methods with Applications for Personal Computers*, Prentice-Hall, Englewood Cliffs, NJ, 1987.
[28] Nelder, J.A. and Mead, R., *Computer Journal*, 7, 308, 1965.
[29] Polak, E., *Computational Methods in Optimization*, Academic Press, New York, NY, 1971.
[30] Henrici, P., *Discrete Variable Methods in Ordinary Differential Equations*, John Wiley and Sons, New York, 1962.
[31] Swerling, P., First order error propagation in a stagewise smoothing procedure for satellite observations, *Journal of Astronautical Science*, 6, 46, 1959.

© 2005 by Chapman & Hall/CRC

[32] Kalman, R.E., A new approach in linear filtering and prediction problems, *Journal of Basic Engineering*, 82, 34, 1960.
[33] Kalman, R.E. and Bucy, R.S., New results in linear filtering and prediction theory, *Journal of Basic Engineering*, 83, 95, 1961.
[34] McGee, A. and Schmidt, S.F., Discovery of the Kalman filter as a practical tool for aerospace and industry, NASA, Ames Research Center, California, 1985.
[35] Sayed, A.H. and Kailath, T., A state-space approach to adaptive RLS filtering, in *IEEE Signal Processing Magazine*, 11(3), 18, 1994.
[36] Tapley, D.B. and Peters, J.G., Sequential estimation using a continuous UDU^T covariance factorization, *Journal of Guidance and Control*, 3(4), 326, 1980.
[37] Schultz, R. *et al.*, Maneuver tracking algorithms for AEW target tracking applications, in *SPIE Conference on Signal Processing and Data Processing of Small Targets*, Orlando, FL, SPIE, 1999.
[38] McIntyre, G.A. and Hintz, K.J., A comparison of several maneuvering target tracking models, in *SPIE Conference on Signal Processing, Sensor Fusion and Target Recognition*, Orlando, FL, SPIE, 1998.
[39] Lee, H. and Tahk, M.-J., Generalized input estimation techniques for tracking maneuvering targets, *IEEE Transactions on Aerospace Electronic Systems*, 35(4), 1388, 1999.
[40] Li, X.R. and Jilkov, V.P., A survey of maneuvering target tracking: dynamic models, in *SPIE Conference on Signal and Data Processing of Small Targets*, Orlando, FL, SPIE, 2000.
[41] Li, X.R. and Jilkov, V.P., A survey of maneuvering target tracking — Part II: Ballistic target models, in *SPIE Conference on Signal and Data Processing of Small Targets*, Orlando, FL, SPIE, 2001.
[42] Li, X.R. and Jilkov, V.P., A survey of manuevering target tracking — Part III: measurement models, in *SPIE Conference on Signal and Data Processing of Small Targets*, Orlando, FL, SPIE, 2001.
[43] Li, X.R. and Jilkov, V.P., A survey of maneuvering target tracking — Part IV: decision-based methods, in *SPIE Conference on Signal and Data Processing of Small Targets*, Orlando, FL, SPIE, 2002.

© 2005 by Chapman & Hall/CRC

19
Data Registration

R.R. Brooks, Jacob Lamb, and Lynne Grewe

19.1 Problem Statement

To fuse two sensor readings, they must be in a common coordinate system. The assumption that the mapping is known *a priori* is unwarranted in many dynamic systems. Finding the correct mapping of one image onto another is known as registration. An image can be thought of as a two-dimensional sensor reading. In this chapter we will provide examples using two- and two-and-a-half-dimensional data. The same approaches can be trivially applied to one-dimensional readings. Their application to data of higher dimensions is limited by the problem of occlusion, where data in the environment are obscured by the relative position of objects in the environment. The first step in fusing multiple sensor readings is registering the images to find the correspondence between them [1].

Existing methods are primarily based on methods used by cartographers. These methods often make assumptions concerning the input data that may not be true. As shown in Figure 19.1, the general problem is: given two N-dimensional sensor readings, find the function F which best maps the reading from sensor two $S_2(x_1, \ldots, x_n)$ onto the reading from sensor one $S_1(x_1, \ldots, x_n)$ so that ideally $F(S_2(x_1, \ldots, x_n)) = S_1(x_1, \ldots, x_n)$. In practice, all sensor readings contain some amount of measurement error or noise, so that the ideal case occurs rarely, if ever.

We will initially approach the registration problem as an attempt to automatically find a gruence (translation and rotation) correctly calibrating two two-dimensional sensor readings with identical geometries. We can make these assumptions without a loss of generality, since:

1. A method for two readings can be sequentially extended to any number of images.
2. Most sensors currently work in one or two dimensions.
3. We presuppose known sensor geometries. If they are known, a function can be derived to map readings as if they were identical.

Most of the work given here finds gruences (translations and rotations), since these functions are representative of the most common problems. Extending these approaches to include the class of affine transformations by adding scaling transformations [3] is straightforward. "Rubber sheet" transformations also exist that warp the contents of the image [4]. The final case study in this chapter transforms images of radically different geometries to a common mapping.

© 2005 by Chapman & Hall/CRC

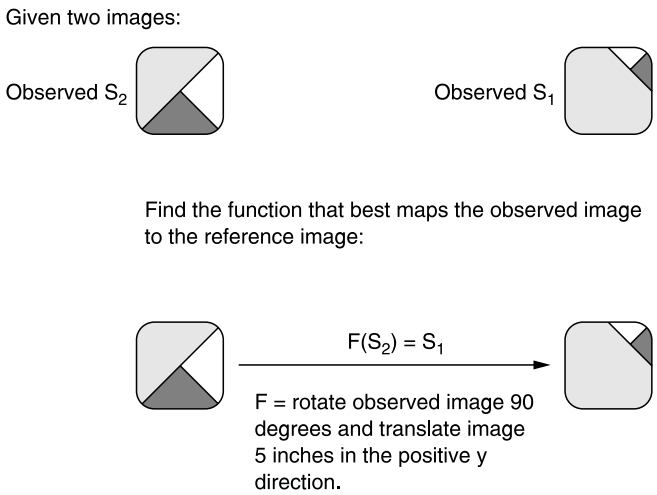


Figure 19.1. Registration is finding the mapping function $F(S_2)$. Adapted from Brooks and Iyengar [2].

19.2 Coordinate Transformations

The math we present for coordinate transformations is based on the concept of rigid body motion. Rigid body motion generally refers to the set of *affine* transformations, which are used to describe either object motion or image transformations. We will refer to the entity transformed as an *image* throughout this section.

Affine transformations are combinations of four elementary operations: rotation, scaling, shearing, and translation. *Rotation* refers to the angular movement of an image. *Scaling* refers to changes in the size of the image. *Shearing* is movement of an image proportional to the distance along an axis. To visualize shearing, imagine a letter printed on a sheet of rubber. If the bottom edge of the sheet of rubber is held in place while the top edge is moved 6 in to the right, the stretching of the sheet of rubber *shears* the image. *Translation* is simple movement of the image in a given direction (e.g. moving an object 4 in to the left).

These transformations can exist in any number of dimensions. Most commonly they are defined in two or three dimensions, describing either an image or an object in space. Often three-dimensional data will be written as *quaternions*, which are four-dimensional vectors. An object with **x** coordinate a, **y** coordinate b, and **z** coordinate c would be represented by the quaternion $(a\ b\ c\ 1)^{\mathrm{T}}$. This formalism simplifies many image operations on images. Quaternions are also useful for the geometry of projections [3,5]. The geometry of projections is outside of the area treated in this book, but is extremely useful for interpreting sensor data.

To illustrate the use of quaternions, we calculate the position $(x\ y\ z)^{\mathrm{T}}$ of point $(a\ b\ c)$ after it has been rotated by θ degrees around the **z** axis and then translated by 14 units in the **y** direction:

$$\begin{vmatrix} x \\ y \\ z \end{vmatrix} = \begin{vmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} a \\ b \\ c \end{vmatrix} + \begin{vmatrix} 0 \\ 14 \\ 0 \end{vmatrix}$$

Using quaternions, this can be represented as a simple matrix multiplication:

$$\begin{vmatrix} x \\ y \\ z \\ 1 \end{vmatrix} = \begin{vmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 14 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} + \begin{vmatrix} a \\ b \\ c \\ 1 \end{vmatrix}$$

© 2005 by Chapman & Hall/CRC

You are advised to calculate the values of x, y, and z for both expressions and verify that they are in fact identical.

We now present the matrices for individual transformations.

$\mathbf{R_x}(\theta)$, rotation about the **x** axis by θ degrees:

$$\begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

$\mathbf{R_y}(\theta)$, rotation about the **y** axis by θ degrees:

$$\begin{vmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

$\mathbf{R_z}(\theta)$, rotation about the **z** axis by θ degrees:

$$\begin{vmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

$\mathbf{T}(t_x, t_y, t_z)$, translation by t_x in the **x** direction, t_y in the **y** direction, and t_z in the **z** direction:

$$\begin{vmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

$\mathbf{S}(s_x, s_y, s_z)$, scaling by s_x in the **x** direction, s_y in the **y** direction, and s_z in the **z** direction:

$$\begin{vmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

A total of six shearing transformations exist, one for each possible combination of the three axes. Since a shear is not technically a rigid body transform, they are omitted from our list; interested readers are referred to Hill [3].

Note that affine transforms preserve linearity, parallelism of lines, and proportional distances [3]. The volume of an object going through an affine transform represented by the matrix **M** is increased by a factor equal to the determinant of **M** [3]:

$$\frac{\text{volume after transform}}{\text{volume before transform}} = \text{determinant}(\mathbf{M})$$

© 2005 by Chapman & Hall/CRC

You can also use matrix multiplication to construct a matrix that will perform any combination of affine transformations. You do so by simply constructing an equation that would perform the transformations sequentially and then calculate the matrix product. Following this logic, translating the vector **v** by 4 in the **z** direction and then rotating it by θ around the **y** axis is $\mathbf{Ry}(\theta)\ \mathbf{T}(0, 0, 4)\mathbf{v}$.

Consider the transformation that takes place when moving a camera C relative to a three-dimensional box B. The camera initially looks at the middle of one side of the box from a distance of 3 m. The boxes vertices are at positions $(2, -1, 3)$, $(2, 1, 3)$, $(-2, -1, 3)$, $(-2, 1, 3)$, $(2, -1, 6)$, $(2, 1, 6)$, $(-2, -1, 6)$, and $(-2, 1, 6)$ relative to the camera. The camera makes the following sequence of motions: the camera moves 10 m further away from the box, pans 20° to the right, and rotates 60° counterclockwise about its axis. We will calculate the new positions of the box's vertices relative to the camera.

Care must be taken that the coordinate system used is consistent throughout the process. In this case the observer is in motion, and all measurements are made in the frame of reference of the observer. For problems of this type, the natural choice of coordinates is called a *viewer-centered* coordinate system. The origin is the center of mass of the observer, and every motion made by the observer is considered as a diametrically opposed motion made by the observed environment. For example, a video camera on a train moving east at 40 miles per hour shows the world moving west at 40 miles per hour.

The transformation matrix for this problem is easily derived. Moving C 10 m away from B is a simple translation. Since C moves -10 m along the z-axis, the matrix **m1** moves $B + 10$ m along the same axis:

$$\begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 10 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Matrix **m2** describes the 20° pan taken by C around B. In essence, this is a three-dimensional rotation of B about the y-axis. The rotation, however, is about the mid-point of B and not the mid-point of C. In order to perform this transformation, the midpoint of C must first be translated to the midpoint of B, the rotation performed, and C must then be translated along the z-axis. Care must be taken to note that rotating C by 20° is seen as rotating B by $-20°$. The matrix **m2** is the result of this matrix multiplication:

$$\begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 13 \\ 0 & 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} 0.94 & 0 & -0.34 & 0 \\ 0 & 1 & 0 & 0 \\ 0.34 & 0 & 0.94 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -13 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

The final transformation is done by rotating C about its own z-axis. The matrix **m3** is therefore a simple rotation about the z-axis of $-60°$:

$$\begin{vmatrix} 0.5 & -0.87 & 0 & 0 \\ -0.87 & 0.5 & 0 & 0 \\ 0 & 0 & 1 & 10 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Again, using these three affine transformations, we can find one single transformation by performing the matrix multiplication $\mathbf{M} = \mathbf{m3}\ \mathbf{m2}\ \mathbf{m1}$. This gives the following equation, which maps any point $(x\ y\ z)$ in the environment to a point $(x'\ y'\ z')$ as now seen by C:

$$\begin{vmatrix} x' \\ y' \\ z' \\ 1 \end{vmatrix} = \begin{vmatrix} 0.47 & -0.87 & -0.17 & 2.21 \\ -0.818 & 0.5 & 0.296 & -3.845 \\ 0.34 & 0 & 0.94 & 0.78 \\ 0 & 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} x \\ y \\ z \\ 1 \end{vmatrix}$$

© 2005 by Chapman & Hall/CRC

This means that the vertices of the cube are: (3.51, −5.102, 4.281), (1.77, −4.093, 4.24), (1.63, −1.821, 2.92), (−0.11, −0.821, 2.92), (−0.11, −0.821, 2.92), (3, −4.205, 7.1), (1.26, −3.205, 7.1), (1.12, −0.933, 5.74), and (−0.62, 0.067, 5.74) after the transformation. C programs are available [2] that perform these transformations for camera images.

19.3 Survey of Registration Techniques

Several methods exist for registering images. Table 19.1 summarizes the features of representative image registration methods discussed in this section. A good survey of this problem is that of Brown [6].

Many processes require that data from one image, called the observed image, be compared with or mapped to another image, called the reference image. Perhaps the largest amount of current image registration research is in the field of medical imaging. One application is sensor fusion to combine outputs from several medical imaging technologies, such as computed tomography and magnetic resonance imaging, to form a more complete image of internal organs [7]. Registered images are then used for medical diagnosis of illness [8], and automated control of radiation therapy [9]. Similar applications of registered and fused images are referred to as using terrain "footprints" in military applications [10], are used in remote sensing applications [11], and are common in robotics as well. A novel application is registering portions of images to estimate motion. Descriptions of motion can then be used to construct intermediate images in television transmissions. Jain and Jain [12] describe the application of this concept to bandwidth reduction in video communications. These are some of the more recent applications relying on accurate image registration, but methods of image registration have been studied since the beginning of the field of cartography.

The traditional method of registering two images is an extension of methods used in cartography. A number of control points are found in both images. The control points are matched, and this match is used to deduce equations that interpolate all points in the new image to corresponding points in the reference image [4,13].

Several algorithms exist for each phase of this process. In various studies, control points have been explicitly placed in the image by the experimenter [9], by edges defined by intensity changes [14], by specific points peculiar to a given image [15], and by line intersections, center of gravity of closed regions, or points of high curvature [13]. Similarly, a large number of methods have been proposed for matching control points in the observed image to the control points in the reference image. The obvious method is to correlate a template of the observed image against the reference image [10,16]. Another widely used approach is to calculate the transformation matrix that describes the mapping with the least-square error [10,13,15]. Other standard computational methods, such as relaxation and hill climbing, have also been used [12,17,15]. Pinz *et al.* [19] use a hill-climbing algorithm to match images and note the difficulty posed by local minima in the search space; to overcome this they ran a number of attempts in parallel with different initial conditions.

Interesting methods have been implemented that consider all possible transformations. Stockman *et al.* [10] construct vectors between all pairs of control points in an image. For each vector in each image, an affine transformation matrix is computed that converts the vector from the observed image to one of the vectors from the reference image. These transformations are then plotted, and the region containing the largest number of correspondences is assumed to contain the correct transformation. Wong and Hall [14] matched scenes by extracting edges or intensity differences and constructing a tree of all possible matches that were below a given error threshold. They reduced the amount of computation needed by stopping all computation concerning a potential matching once the error threshold had been exceeded, but this method is nonetheless computationally intensive.

Registration of multi-sensor data to a three-dimensional scene given *a priori* knowledge of the contents of the scene is discussed Chellappa *et al.* [20]. The use of an extended Kalman filter to register moving sensors in a sensor fusion problem is discussed by Zhou *et al.* [21].

A number of researchers have used multi-resolution methods to prune the search space considered by their algorithms. Mandara and Fitzpatrick [8] use a multi-resolution approach to reduce the size of

© 2005 by Chapman & Hall/CRC

Table 19.1. Image registration methods

Algorithm	Image type	Matching method	Interpolation function	Transforms supported	Comments
Andrus	Boundary maps	Correlation	None	Gruence	Noise intolerant, small rotations
Barnea	No restriction	Improved correlation	None	Translation	No rotation, scaling noise, rubber sheet
Barrow	No restriction	Hill climbing	Parametric chamfer	Gruence	Noise intolerant, small displacement
Brooks Iyengar	No restriction	Elitist genetic algorithm	None	Gruence	Noise tolerant, tolerates periodicity
Cox	Line segments	Hill climbing	None	Gruence	Matches using small number of features
Davis	Specific shapes	Relaxation	None	Affine	Matches shapes
Goshtasby 1986	Control points	Various	Piecewise linear	Rubber sheet	Fits images using mapped points
Goshtasby 1987	Control points	Various	Piecewise cubic	Rubber sheet	Fits images using mapped points
Goshtasby 1988	Control points	Various	Least squares	Rubber sheet	Fits images using mapped points
Jain	Sub-images	Hill climbing	None	Translation	Small translations No rotation, no noise
Mandara	Control points	Classic G.A S.A.	Bi-linear	Rubber sheet	Fits 4 fixed points using error fitness
Mitiche	Control points	Least squares	None	Affine	Uses control points
Oghabian	Control points	Sequential search	Least squares	Rubber sheet	Assumes small displacement
Pinz	Control points	Tree search	None	Affine	Difficulty with local minima
Stockman	Control points	Cluster	None	Affine	Assumes landmarks, periodicity problem
Wong	Intensity differences	Exhaustive search	None	Affine	Uses edges, intense computation

© 2005 by Chapman & Hall/CRC

their initial search space for registering medical images using simulated annealing and genetic algorithms. This work influenced Oghabian and Todd-Pokropek [7], who similarly reduced their search space when registering brain images with small displacements. Pinz *et al.* [19] adjusted both multi-resolution scale space and step size in order to reduce the computational complexity of a hill-climbing registration method. By starting with low-resolution images, these researchers believe that rejecting large numbers of possible matches is possible, and that the correct match can be found by progressively increasing the resolution. Note that, in images with a strong periodic component, a number of low-resolution matches may be feasible. In which case, the multi-resolution approach will be unable to prune the search space and instead will increase the computational load.

The approach of Grewe and Brooks [22], which we discuss in more detail in Section 19.6, was to use a multi-resolution technique, the wavelet transform, to extract features used to register images. Others have also applied wavelets to this problem, including using locally maximum wavelet coefficient values as features from two images [23]. The centroids of these features are used to compute the translation offset between the two images. This use of a simple centroid difference is subject to difficulties when the scenes only partially overlap and hence contain many other different features. A principle components analysis is then performed and the eigenvectors of the covariance matrix provide an orthogonal reference system for computing the rotation between the two images. In another example [24], the wavelet transform is used to obtain a complexity index for two images. The complexity measure is used to determine the amount of compression appropriate for the image. Compression is then performed, giving a small number of control points. Images made up of control points for rotations are then tested to determine the best fit.

19.4 Objective Functions

In optimization literature, the function to be optimized is often called the *objective function* [25–27]. The term "fitness function" is used in genetic algorithms literature [2,28,29] in a manner similar to the use of objective functions in optimization literature. Mappings can be defined by affine transformations. The terms objective function and fitness function will be used interchangeably in this chapter. Here, we present a few functions that are appropriate for data registration. The proper function to use is dependent on the amount and type of noise present in the data.

If the noise is approximately Gaussian (i.e. white noise), then it follows a normal distribution and has an expected value of zero. A fitness function can be derived by first computing the intersection between the two image sensors, sensor 1 and sensor 2, using the proposed mapping function. The gray levels of every pixel from sensor 1 in the intersection are compared with the gray level of the corresponding sensor-2 pixel. We define $\text{read}_1(x, y)$ as the value returned by sensor 1 at point (x, y), and $\text{read}_2(x', y')$ as the reading returned by sensor 2 at point (x', y'). Point (x', y') is found by reversing the translation and rotation defined by the parameters being tested. You can present the difference of $\text{read}_1(x, y)$ and $\text{read}_2(x', y')$ as

$$\text{read}_1(x,y) - \text{read}_2(x',y') = [v_1(x,y) + \text{noise}_1(x,y) - v_2(x',y') + \text{noise}_2(x',y')] \tag{19.1}$$

where $v_1(x, y)$ and $v_2(x', y')$ are the actual gray-scale values, and $\text{noise}_1(x, y)$ and $\text{noise}_2(x, y)$ are the noise in the sensor-1 and sensor-2 readings respectively. This expression can be rewritten as

$$\text{read}_1(x,y) - \text{read}_2(x',y') = [v_1(x,y) - \text{v}_2(x',y')] + [\text{noise}_1(x,y) - \text{noise}_2(x',y')] \tag{19.2}$$

If we square this value and sum it over the entire intersection, we get

$$\sum [\text{read}_1(x,y) - \text{read}_2(x',y')]^2 = \sum \{[v_1(x,y) - v_2(x',y')]^2 + [\text{noise}_1(x,y) - \text{noise}_2(x',y')]\}^2 \tag{19.3}$$

© 2005 by Chapman & Hall/CRC

Note that when the parameters are correct, the gray-scale values $v_1(x,y)$ and $v_2(x',y')$ will be identical, and this expression becomes:

$$\sum [\text{read}_1(x,y) - \text{read}_2(x',y')]^2 = \sum [\text{noise}_1(x,y) - \text{noise}_2(x',y')]^2 \tag{19.4}$$

Since all noise follows the same distribution with the same variance, the expected value of this is identical for all intersections of the same area and, as such, the minimum value for the function over all intersections of a given area. Variation in this value thus consists of two parts: the difference in the gray-scale values of the noise-free image, and a random factor that is distributed according to a chi-square distribution of unknown variance. The number of degrees-of-freedom for the chi-square distribution is the number of pixels in the intersection.

We can have small intersections that match coincidentally. In order to favor intersections of larger area, we divide by the square of the number of pixels in the intersection. The fitness function thus becomes

$$\frac{\sum [\text{read}_1(x,y) - \text{read}_2(x',y')]^2}{(\text{Number of pixels in the intersection})^2} \tag{19.5}$$

The expected value of a chi-square function is the number of degrees of freedom, and the number of degrees of freedom in this case is equal to the number of pixels in the intersection. In the case of a perfect fit (i.e. $v_1(x, y) = v_2(x', y')$), the expected value of this function is therefore within a constant factor of

$$\frac{1}{\text{Number of pixels in the intersection}} \tag{19.6}$$

This function is the summation of the error per pixel squared over the intersection of the sensor-1 and sensor-2 readings. As shown above, the unique global minimum of this function is found when using the parameters that define the largest intersection where the gray-scale values of sensor 1 are the same as the gray-scale values of the translated and rotated sensor-2 reading. The fitness function is thus

$$\frac{\sum [\text{read}_1(x,y) - \text{read}_2(x',y')]^2}{K(W)^2} = \frac{\sum \{[\text{gray}_1(x,y) - \text{gray}_2(x',y')] + [\text{noise}_1(x,y) - \text{noise}_2(x',y')]^2\}}{K(W)^2} \tag{19.7}$$

where: w is a point in the search space; $K(w)$ is the number of pixels in the overlap, $w(x', y')$ is the point corresponding to (x, y) for $\text{read}_1(x, y)$; $\text{read}_1(x, y)$ ($\text{read}_2(x',y')$) is the pixel value from sensor 1 (2) at point (x, y) $((x', y'))$; $\text{gray}_1(x, y)$ ($\text{gray}_2(x',y')$) is the noiseless value for sensor 1 (2) at (x, y) $((x', y'))$; $\text{noise}_1(x, y)$ ($\text{noise}_2(x', y')$) is the noise in the sensor-1 (-2) reading at (x, y) $((x', y'))$.

This function has been shown to reflect the problem adequately when the noise at each pixel follows a Gaussian distribution of uniform variance. In our experiments this is not strictly true, when the gray scale is limited to 256 discrete values. Because of this, when the gray-scale value is 0 (255) the noise is limited to positive (negative) values. For large intersections, however, this factor is not significant.

Other noise models can be accounted for by simply modifying the fitness function. Another common noise model is the salt-and-pepper noise. Malfunctioning pixels in electronic cameras or dust in optical systems commonly cause this type of noise. In this model, the correct gray-scale value in a picture is

© 2005 by Chapman & Hall/CRC

replaced by a value of 0 (255) with an unknown probability $p(q)$. An appropriate fitness function for this type of noise is Equation (19.8):

$$\sum_{\substack{\text{read}_1(x,y)\neq 0\\ \text{read}_1(x,y)\neq 255\\ \text{read}_2(x',y')\neq 0\\ \text{read}_2(x',y')\neq 255}} \frac{[\text{read}_1(x,y)-\text{read}_2(x',y')]^2}{K(W)} \tag{19.8}$$

A similar function can be derived for uniform noise by using the expected value $E[(U_1-U_2)^2]$ of the squared difference of two uniform variables U_1 and U_2. An appropriate fitness function is then given by Equation (19.9):

$$\sum \frac{[\text{read}_1(x,y)-\text{read}_2(x',y')]^2}{E\left[(U_1-U_2)^2\right]K(w)} \tag{19.9}$$

19.5 Results from Meta-Heuristic Approaches

In this section we consider two-dimensional images, without occlusion and projection, although the results in this section could easily be extended to account for projection. We consider two images, which can be matched using gruence (translation and rotation) transformations. Addition of scaling would make the class of transformations include all affine transformations. Occlusion and projection require knowledge of the scene structure and interpretation of the sensor data before registration. It is doubtful that is feasible in real time.

Given two noisy overlapping sensor readings, compute the optimal gruence (i.e. translation and rotation) mapping one to the other. The sensors return two-dimensional gray-level data from the same environment. Both sensors have identical geometric characteristics. They cover circular regions such that the two readings overlap. Since the size, position, and orientation of the overlaps are unknown, traditional image-processing techniques are unsuited to solving the problem. For example, the method of using moments of a region is useless in this context [31].

Readings from both sensors are corrupted with noise. In Section 19.4 we derived appropriate fitness (objective) functions for different noise models. The Gaussian noise model is applicable to a large number of real-world problems. It is also the limiting case when a large number of independent sources of error exist.

This section reviews results from Brooks and co-workers [2,30] where we attempted to find the optimal parameters (x_T, y_T, θ) defining the relative position and orientation of the two sensor readings. The search space is a three-dimensional vector space defined by these parameters, where a point is denoted by the vector $w=[x_T, y_T, \theta]^{\mathrm{T}}$. Figure 19.2 shows the transformation between the two images given in Equation (19.10):

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & x_T \\ \sin\theta & \cos\theta & x_T \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{19.10}$$

Figure 19.3 shows the artificial terrain used in the experiments described by Brooks and co-workers [2,30]. It contains significant periodic and nonperiodic components. Two overlapping circular regions are taken from the terrain. One of them is rotated. These are virtual sensor readings. The readings are also corrupted with noise. The experiments use optimization methods to search for the proper mapping

© 2005 by Chapman & Hall/CRC

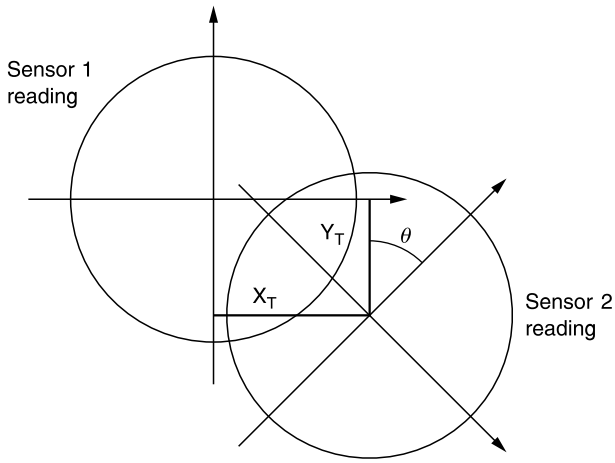


Figure 19.2. Geometric relationship of two sensor readings. From Chen *et al.* [30].

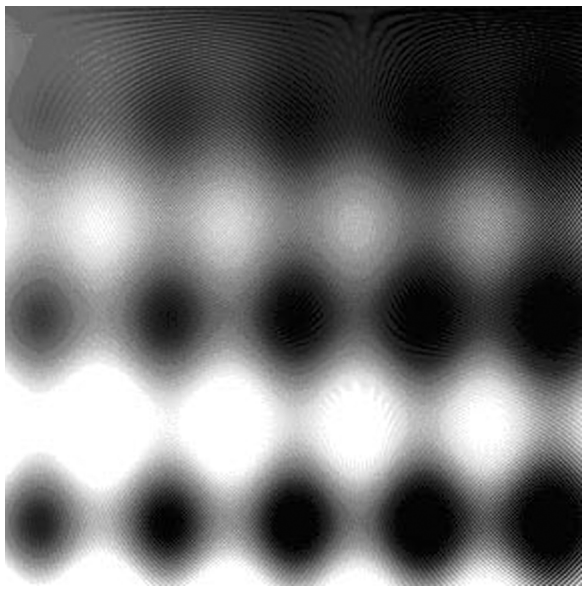


Figure 19.3. Terrain model.

of sensor 1 to sensor 2. Figures 19.4 and 19.5 give examples of the two sensor readings used with different noise levels.

Figure 19.6 shows paths taken by the tabu search algorithm when searching for an optimal match between the two sensor readings with increasing noise. The search starts in the middle of the sensor-1 reading. Note that the correct answer would have been at the bottom right-hand edge of the sensor-2 reading. The left-hand image shows the path taken during 75 iterations when the error variance was increased to 90. Note that, even in the presence of noise that is strong enough to obscure most of the information contained in the picture, the search took approximately the same path as with very little noise.

Figure 19.7 shows gene pool values after a number of iterations of a genetic algorithm using a classic reproduction scheme with variance values of 1 (right) and 90 (left). This figure shows that, even in the

© 2005 by Chapman & Hall/CRC

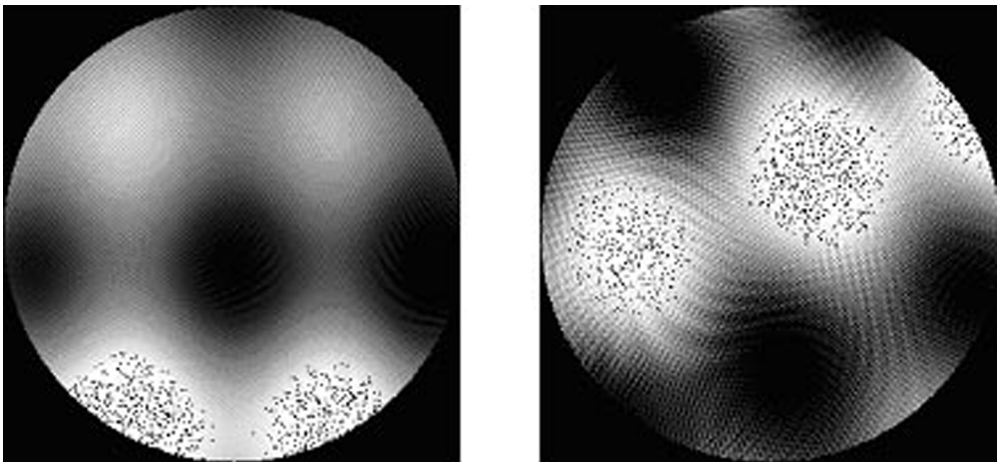


Figure 19.4. Sensor-1 reading with noise variance of 1 (left). Sensor-2 reading with noise variance of 1 (right).

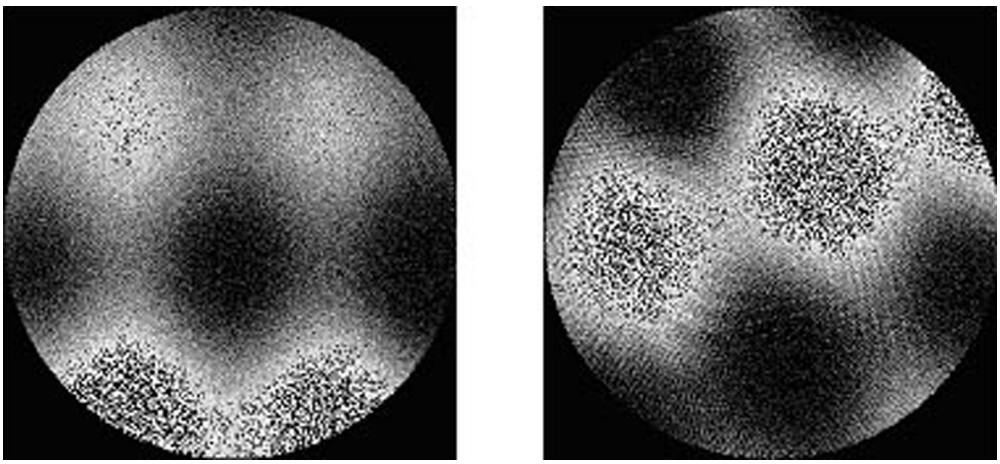


Figure 19.5. Sensor-1 reading with noise variance of 33 (left). Sensor-2 reading with noise variance of 33 (right).

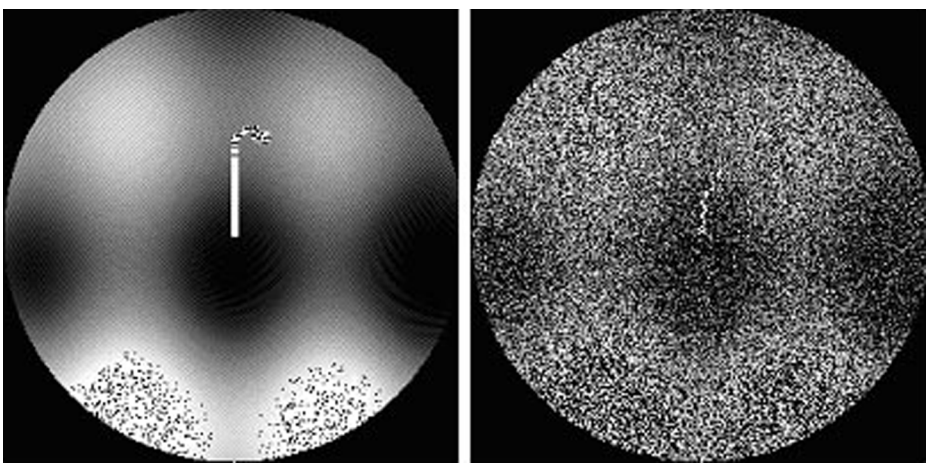


Figure 19.6. Search path taken by tabu search registration method. Noise increases on left.

© 2005 by Chapman & Hall/CRC

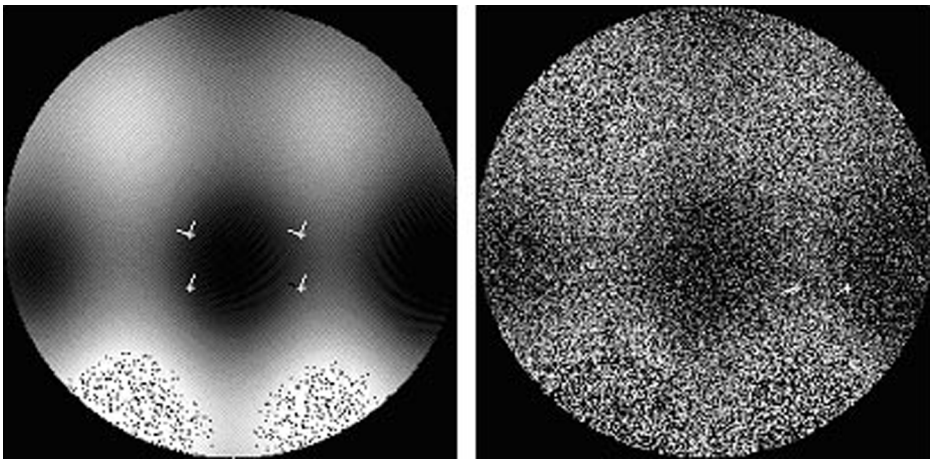


Figure 19.7. Gene pools from classic genetic algorithm seeking image mapping parameters.

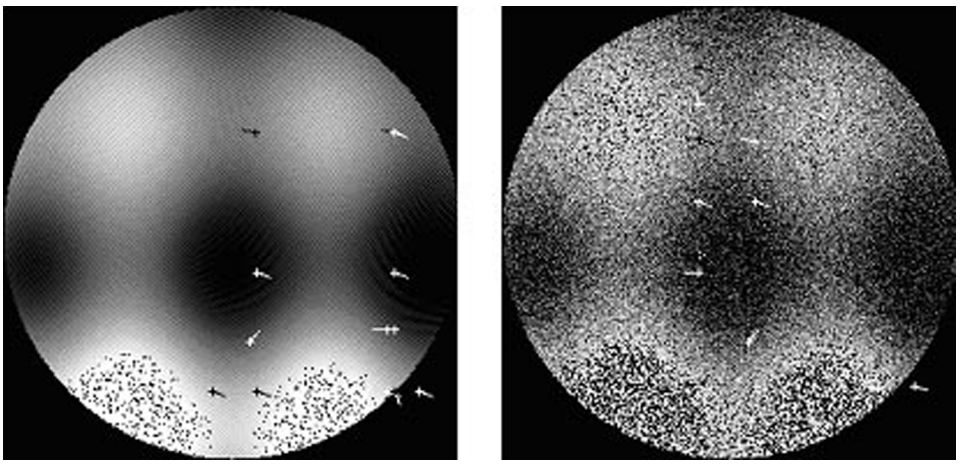


Figure 19.8. Gene pools from elite genetic algorithm.

presence of noise, the values contained in the gene pool tend to converge. Unfortunately, convergence is not to globally optimal values.

Figure 19.8 shows gene pools found by the elite reproduction scheme after several iterations. Notice that the images contain values very close to the globally optimal value (the values near the lower left edge of the sensor reading). The genetic algorithm with the elite reproduction scheme tended to converge towards the globally optimal value even in the presence of moderate noise.

The elitist approach converged rapidly towards good solutions to the problem. In fact, the shapes of the graphs are surprisingly similar considering the differences in the images they are treating. In spite of the fact that the algorithm converged towards good solutions even in the presence of overwhelming amounts of noise, a limit existed to its ability to find the globally optimal solution. Note that the globally optimal parameter values are x displacement $= 91$, y displacement $= 91$, rotation $=$ 2.74889 radians. The algorithm does not always find the globally optimal values, but tends to do a good job even in the presence of moderate amounts of noise. However, once the noise reaches a point where it obscures too much of the information present in the image, it no longer locates the optimal values.

© 2005 by Chapman & Hall/CRC

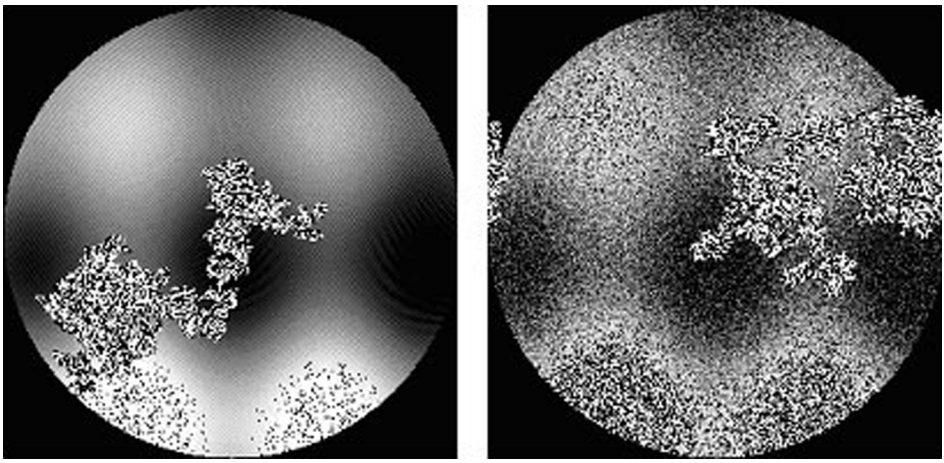


Figure 19.9. Search paths taken by simulated annealing.

Simulated annealing and TRUST are the methods used with clear termination criteria. This makes a direct comparison with tabu search and genetic algorithms difficult. The final answers found by simulated annealing were roughly comparable to the answers found by tabu search. The number of iterations used to find these answers was much larger than the number of iterations taken by tabu search. Figure 19.9 displays the paths taken by simulated annealing searching for the correct registration in the presence of varying noise levels. Since the correct answer is in the lower right-hand corner of the sensor reading, obviously simulated annealing did not converge to the global optimum. The simulated annealing approach searched within any of a number of local optima. It did not remain trapped in the region defined by the first local optima, but it did not find the global optimum either.

Figure 19.10 shows paths taken by TRUST when the global optimum is in the upper right-hand corner with a rotation of 2.49 radians. The line from the center (0, 0) goes to bound (255, 255) and the search stops. This illustrates TRUST's ability to find local minima and quickly climb out of their basins of attraction. After 300 iterations, the results of tabu search are not even close to the global minimum. The results from TRUST and elite genetic algorithms are almost identical, except TRUST has concrete stopping criteria. It has finished examining the search space. We can be more certain that a global minimum has been located. We tested TRUST using noise with seven different variances under different

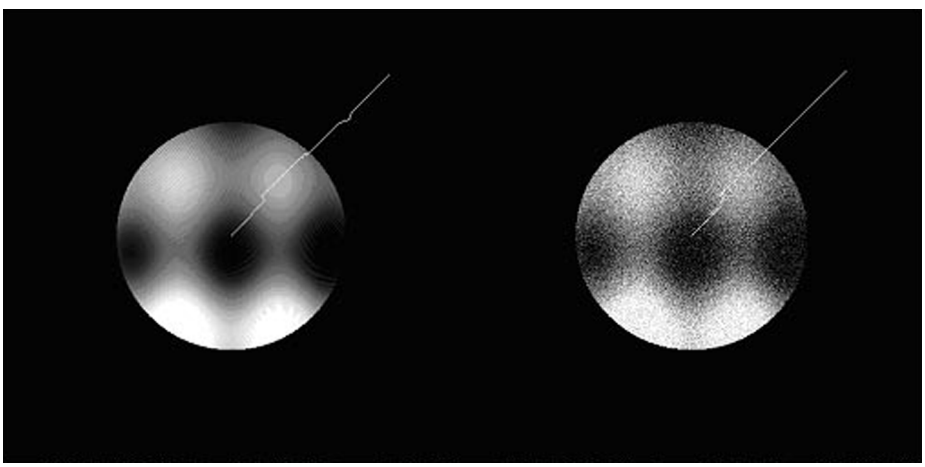


Figure 19.10. Paths taken by TRUST with noise variance 0.0 (left) and 30.0 (right).

© 2005 by Chapman & Hall/CRC

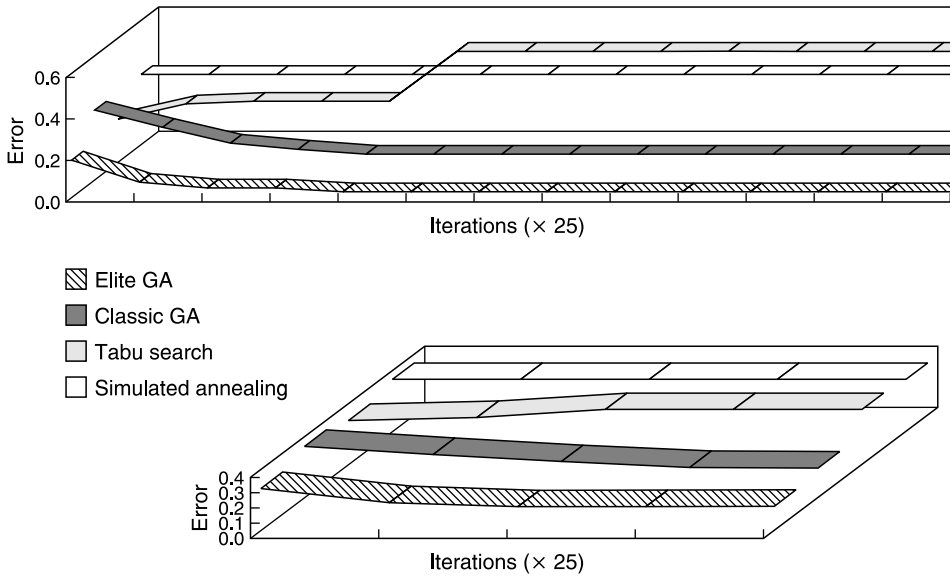


Figure 19.11. Fitness function results variance 1 (top) and 50 (bottom).

conditions and compare the results with the elitist genetic algorithm. Both the elitist genetic algorithm and TRUST can handle noise with a variance of up to 30. The algorithms do not always find the global minima, but the TRUST results show the optimal value can be found even in the presence of large amounts of noise. When the noise reaches levels such as 70, or 90, it obscures the images and it becomes impossible to find the correct answer.

Figures 19.11 and 19.12 show the value of the best parameter set found by the approaches for two separate sets of experiments. Tabu search, elite genetic algorithms and TRUST tend to move towards locally optimal values and are stable in manner. They show the relationship of the fitness function value of the best parameter set to the number of iterations used by the algorithm. In Figure 19.11, the best answer found by simulated annealing is represented by a straight line. This is not an entirely fair representation. The first several iterations of the algorithm are when the temperature parameter is at its

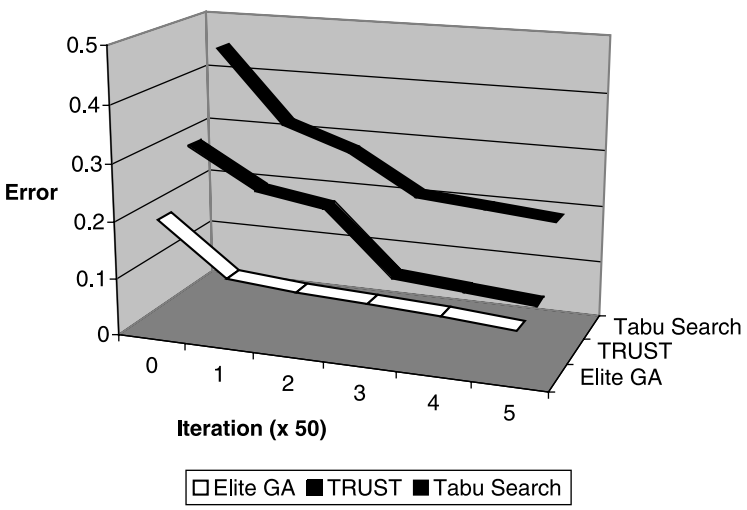


Figure 19.12. Average fitness function value for elite genetic algorithm, TRUST, and tabu search.

© 2005 by Chapman & Hall/CRC

highest point, in which case the algorithm amounts to a random walk. This is intentional; convergence is delayed until a later point in the algorithm when the system starts to cool.

19.6 Feature Selection

Figure 19.13 shows a block diagram of the WaveReg system from Grewe and Brooks [22]. The registration process begins with the transformation of range image data to the wavelet domain. Registration may be done using one decomposition level of this space to greatly reduce the computational complexity of registration. Alternatively, hierarchical registration across multiple levels may be performed. Features are extracted from the user-selected wavelet decomposition level. Users also determine the amount of compression desired in this level. Matches between features from the two range images are used to hypothesize transformations between the two images, which are then evaluated. The "best" transformations are retained.

Figure 19.14(a) shows the main user interface. The user can select to perform registration from an initial estimate if one is known. Other options can be altered from defaults in the Options Selection Box shown in Figure 19.14(b). Features include data compression thresholds, the wavelet level for use in registration, and the number of features to use.

The registration process starts by applying a Daubechies-4 wavelet transform to each range image. The Daubechies-4 wavelet was chosen for compactness. Wavelet data are compressed by thresholding to eliminate low-magnitude wavelet coefficients. The wavelet transform produces a series of three-dimensional edge maps at different resolutions. Maximal wavelet values indicate a sharp change in depth. Figure 19.15 illustrates this. Figure 19.15(a) is the original range map. Figure 19.15(b) is the corresponding intensity image of the human skull. Figure 19.15(c) is the resulting wavelet transform. Figure 19.15(d) is a blow-up of one decomposition level (one level of resolution). Note how maximum values in Figure 19.15(d) correspond to regions where large depth changes occur. The edges may be due to object boundaries, often called jump edges, or may be due to physical edges/transitions on the object, called roof edges.

Features, "special points of interest" in the wavelet domain, are simply points of maximum value in the wavelet decomposition level under examination. They are selected so that no two points are close to each other. Users can specify the distance, or a default value is used. The distance is scaled consistent

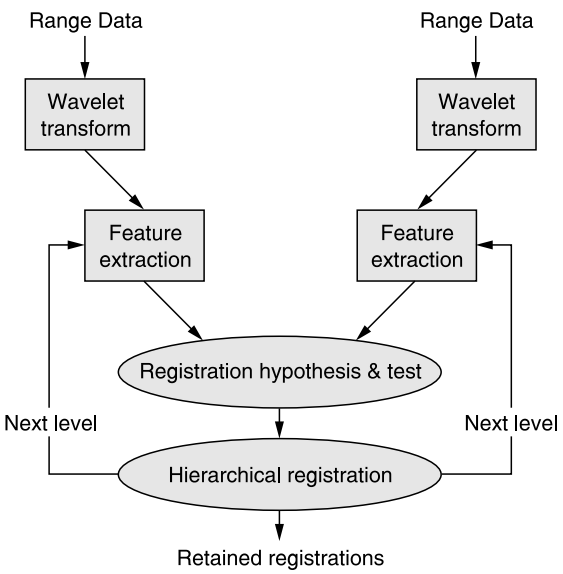


Figure 19.13. Block diagram of WaveReg system.

© 2005 by Chapman & Hall/CRC

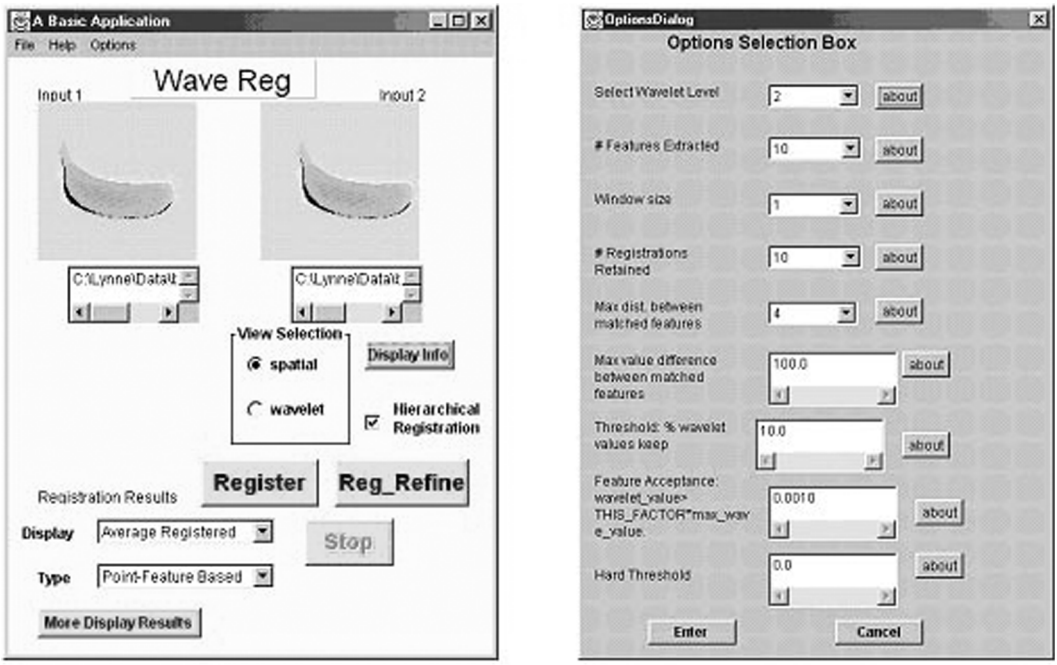


Figure 19.14. (a) Main interface of system. (b) Set of user options.

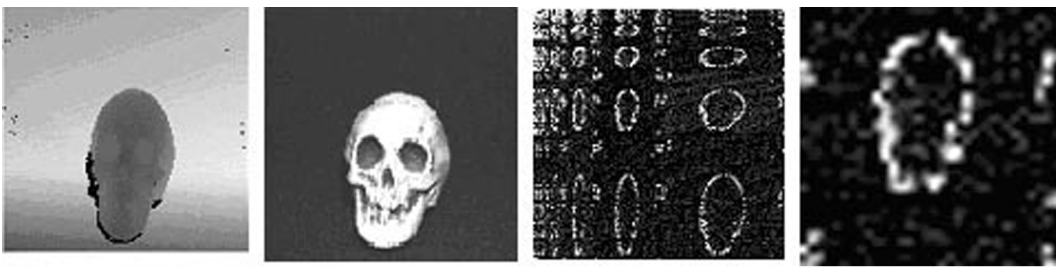


Figure 19.15. (a) Range image. (b) Corresponding intensity image. (c) Wavelet space. (d) Blow-up of level 2 of wavelet decomposition space (originally 32×32 pixels).

with the wavelet decomposition level under examination. Users may specify the number of features to extract at the first decomposition level. For hierarchical registration, this parameter is appropriately scaled for each level. Thresholds can also be changed from their defaults to influence the number of features extracted at each wavelet level. We found that the empirical defaults work well for a range of scenes. Figure 19.16 shows features detected for different range scenes at different wavelet levels. Notice how they correspond to sharp changes in depth.

The next step is verifying possible correspondences between features extracted from the unregistered range images. Each such hypothesis is a possible registration and is evaluated for goodness of fit. The best fits are retained. Hypothesis formation begins at the user-selected wavelet decomposition level L. The default value for L is 2. If hierarchical registration is performed, then registrations retained at level L are refined at level $L-1$. The process continues until we reach the lowest level in the wavelet space.

© 2005 by Chapman & Hall/CRC

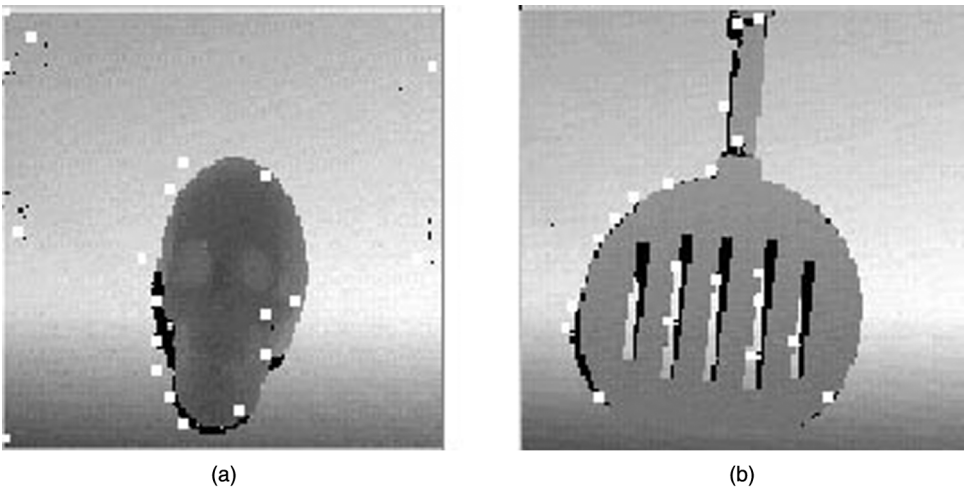


Figure 19.16. Features detected; approximate location indicated by white squares. (a) For wavelet level 2. (b) For wavelet level 1.

For each hypothesis, the transformation for matching the features is calculated and the features from one range image transformed to the other's coordinate space. This reduces the number of computations needed for hypothesis evaluation compared with nonfeature-based registration like Section 19.5. Features are compared and they match if they are close in value and location. Hypotheses are ranked by the number of features matched and how closely their values match. Figure 19.17 illustrates registration results on a range of scenes. These results are restricted to translations.

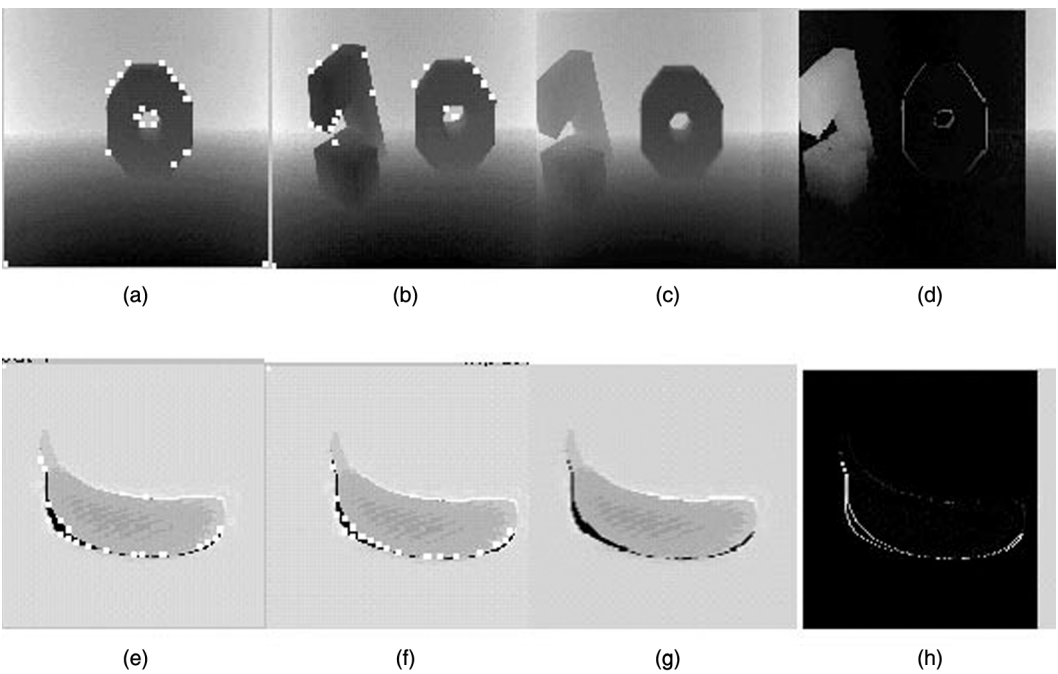


Figure 19.17. (a) Features extracted level 1, image 1. (b) Features extracted level 1, image 2. (c) Merged via averaging registered images. (d) Merged via subtraction of registered images. Parts e–h use the same process but displays different image.

© 2005 by Chapman & Hall/CRC

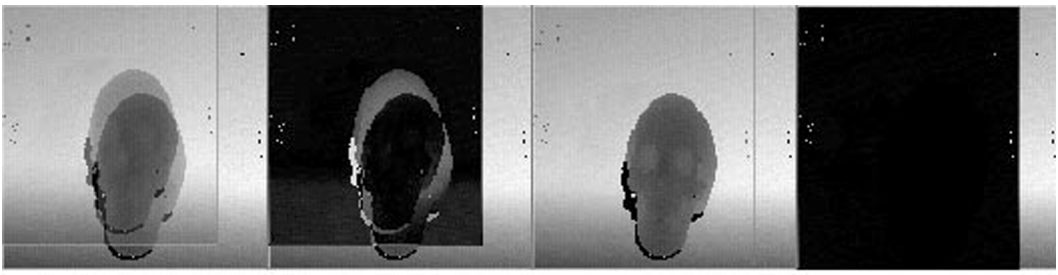


Figure 19.18. Correction of registration by hierarchical registration. (a) Incorrect registration, level 2, merged via averaging. (b) Same incorrect registration merged via subtraction. (c) Correct registrations retained after refined registration at level 1, merged via averaging. (d) Same as (a), merged via subtraction.

For an $N \times N$ image, the number of features extracted is typically $O(c)$. The search algorithm involves $O(c^2) = O(c)$ iterations. Compare this with a data-based approach which must compare $O(N^2)$ points for $O(N^2)$ iterations. Compared with other feature-based approaches, the compactness of the wavelet domain and its multi-resolution nature reduce the number of features extracted. Hierarchical registration also allows efficient refinement of registration at higher resolution, yielding progressively more accurate results.

We tested this approach on several scenes. The system successfully found the correct registration in a retained set of ten registrations for all scenes. Usually, only correct registrations are retained. Figure 19.17 shows images being registered and the results. Both averages and subtractions of registered images are shown.

One interesting point is that the location of an object in a scene can significantly change the underlying wavelet values [32]. Values will still be of the same order of magnitude, but direct comparison of values can be problematic, expecially at higher levels of decomposition. It may be necessary to perform registration only at low decomposition levels. One way to resolve this is by using hierarchical registration. This usually eliminates incorrect registrations through further refinement at lower wavelet levels. Figure 19.18 illustrates this situation where in (a) an incorrect registration is retained at wavelet level 2 that is rejected in (b) during refined registration at wavelet level 1 and replaced by the correct registration.

When an image contains strong periodic components, our feature points may not define a unique mapping of the observed image to the reference image. This can result in incorrect registrations. Note that this is a problem with any feature-based registration approach, and also can cause problems for simple correlation-type registration systems [2].

19.7 Real-Time Registration of Video Streams with Different Geometries

This section discusses a method for registering planar video images to images from a camera with a 360° field of view in real time. First, we discuss three primary concepts vital to the implementation of the fast alignment algorithm. Covered first is catadioptric sensor geometry, which is used to obtain omni images. Then we cover featureless alignment of planar images and provide an algorithm proposed by Picard and Mann to accomplish such a goal. Finally, we describe the CAMSHIFT algorithm as it is used in a face-tracking application.

The limited fields of view inherent in conventional cameras restrict their application. Researchers have investigated methods for increasing field of view for years [33,34]. Recently, there have been a number of implementations of cameras that use conic mirrors to address this issue [33]. Catadioptric systems combine refractive and reflective elements in a vision system [35]. The properties of these

© 2005 by Chapman & Hall/CRC

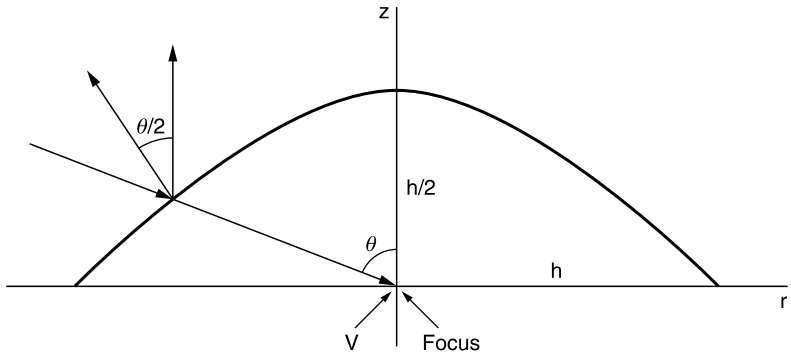


Figure 19.19. Mirror cross-section.

systems have been well studied in literature on telescopes. The first panoramic field camera was proposed by Rees [34]. Recently, Nayar and Peri [36] proposed a compact catadioptric imaging system using folded geometry that provides an effective reflecting surface geometrically equivalent to a paraboloid. This system uses orthographic projection rather than perspective projection.

Cameras providing a field of view greater than conventional cameras are useful in many applications. A simple but effective mechanism for providing this functionality is the placement of a paraboloid mirror immediately in front of the camera lens. Though this solution would provide information about an expanded portion of the environment, constraints need to be applied for the information to be easily used. If an omnidirectional imaging system could view the world in $360^\circ \times 180^\circ$ from a single effective pinhole, then planar projection images can be generated from the omni image by projecting the omni image onto a plane [33]. Similarly, panoramic images can be constructed from an omni image by projecting the omni image onto the inside surface of a cylinder. Figure 19.19 shows how a paraboloid mirror can be used to expand field of view.

The cross-section of the reflecting surface is given by $z(r)$. The cross-section is then rotated about the z axis to give a solid of revolution, the mirror surface. The viewpoint v is at the focus of the paraboloid. Light rays from the scene headed in the direction of v are reflected by the mirror in the direction of the orthographic projection. The angle θ gives the relation between the incoming ray and $z(r)$, then

$$\tan\theta = \frac{r}{z} \tag{19.11}$$

The surface of the mirror is specular, making the angles of incidence and reflectance $\theta/2$. The slope of the mirror surface at the point of reflectance is

$$\frac{\mathrm{d}z}{\mathrm{d}r} = \tan\frac{\theta}{2} \tag{19.12}$$

Substituting the trigonometric identity

$$\tan\theta \frac{2\tan\frac{\theta}{2}}{1-\tan^2\frac{\theta}{2}} \tag{19.13}$$

yields

$$\frac{-2(\mathrm{d}z/\mathrm{d}r)}{1-(\mathrm{d}z/\mathrm{d}r)^2} = \frac{r}{z} \tag{19.14}$$

© 2005 by Chapman & Hall/CRC

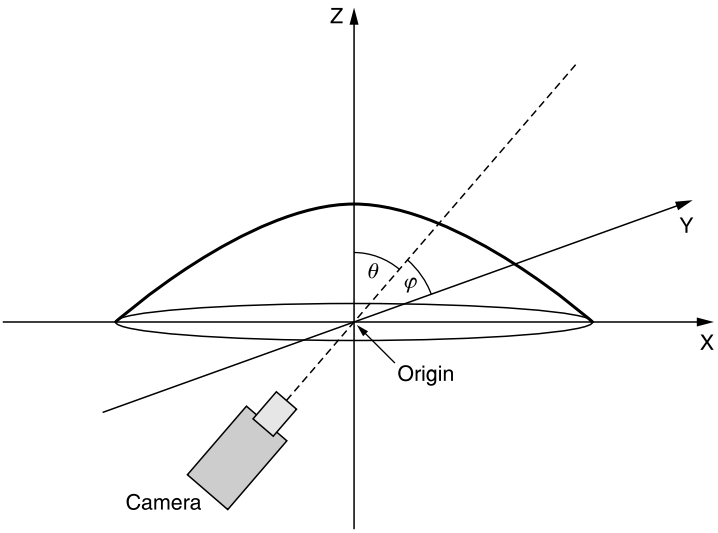


Figure 19.20. Projection geometry.

which indicates that the reflecting surface must satisfy a quadratic first-order differential equation [33]. This can be solved to give the following equation for a reflecting surface that guarantees a single effective viewpoint.

$$z = \frac{h^2 + r^2}{2h} \tag{19.15}$$

This is a paraboloid, with h being the radius of the paraboloid at $z = 0$. The distance between the vertex and its focus is $h/2$. If the paraboloid is terminated at $z = 0$, then the field of view equals exactly one hemisphere. A virtual planar image may be created by placing a plane at a tilt angle θ and pan angle ϕ and projecting the omni image onto the plane. Figure 19.20 shows the geometry of this projection.

The world coordinate system is represented by $\boldsymbol{X}$, $\boldsymbol{Y}$, and $\mathbf{Z}$ vectors, with the origin located at O. The equation for the mirror is related to the world coordinates as

$$Z = \frac{h^2 + r^2}{2h} \tag{19.16}$$

where $r = X^2 + Y^2$ and $h/2$ is the focal distance of the mirror.

The tilt angle is measured clockwise from the $\boldsymbol{Z}$ axis and the pan angle is measured counter-clockwise from the $\boldsymbol{X}$ axis. Given a line of sight from the origin to a point in the scene (x_p, y_p, z_p) at a distance ρ from its focus, where

$$\rho = \frac{h}{(1 + \cos\theta)} \tag{19.17}$$

the projection of the omnidirectional image pixel at (θ, ϕ) onto the perspective plane is then given by

$$x = \rho \sin\theta \cos\phi, \quad y = \rho \sin\theta \sin\phi \tag{19.18}$$

© 2005 by Chapman & Hall/CRC

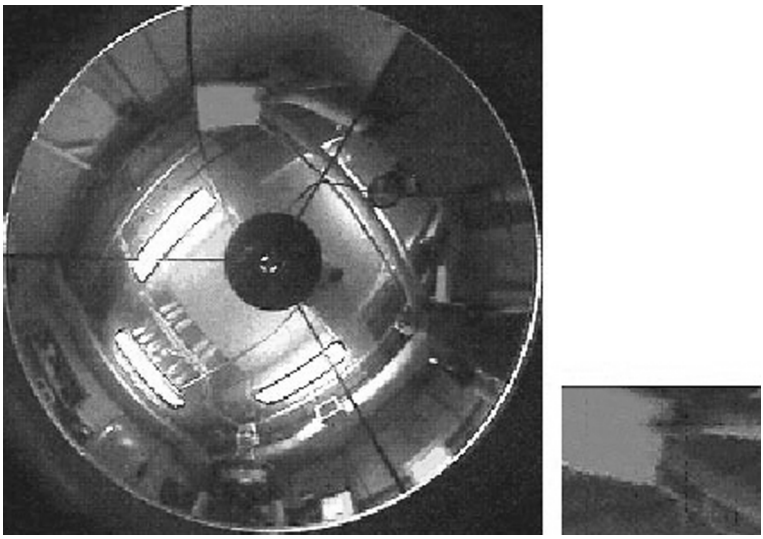


Figure 19.21. Omni and virtual planar images.

Figure 19.21 shows an example of an omni image and a virtual perspective image derived from the omni image.

Mann and Picard [37] present a featureless alignment of images method. They prove that an eight-parameter projective transformation is capable of exactly describing the motion of a camera relative to a scene. Their algorithm is robust under noisy conditions, and applicable in scenes with varied depth. It is suited for the alignment needed to provide the initial bounding area for tracking.

Small changes from one image to another can be measured with optical flow. The optical flow equation assumes that every point $\mathbf{x}$, where $\mathbf{x} = (x, y)^{\mathrm{T}}$, in a frame t is a translated version of some point $\mathbf{x} + \Delta\mathbf{x}$ in frame $t + \Delta t$. The optical flow velocity $\mu_{\mathrm{f}} = (u, v)$ at point $\mathbf{x}$ is $\Delta\mathbf{x}/\Delta t$. If $\mathbf{E_t}$ and $\mathbf{E_x} = (E_x, E_y)$ are the temporal and spatial derivatives of the frame, then

$$\mu_{\mathrm{f}}^{\mathrm{T}}\mathbf{E_x} + \mathbf{E_t} \approx 0 \tag{19.19}$$

gives the traditional optical flow equation.

Given the optical flow between two frames, g and h, a projective transformation that aligns g with h can be found in the following manner. Represent the coordinates of a pixel in g as $\mathbf{x}$ and the coordinates of the corresponding pixel in h as $\mathbf{x}'$. The coordinate transformation from $\mathbf{x}$ to $\mathbf{x}'$ is given as

$$\mathbf{x}' = \left[x', y'\right]^{\mathrm{T}} = \frac{A[x, y]^{\mathrm{T}} + b}{c^{\mathrm{T}}\left[x, y\right]^{\mathrm{T}} + 1} \tag{19.20}$$

where the eight parameters of the transformation are given by $p = [A,\ b,\ c,\ 1]$, $A \in \Re^{2\times 2}, b \in \Re^{2\times 1}, c \in \Re^{2\times 1}$. The model velocity μ_{m} is then given as $\mu_{\mathrm{m}} = \mathbf{x}' - \mathbf{x}$. There will be some discrepancy between the model and flow velocities due to errors in flow calculation and other errors in the model assumption. The error between the two can be defined as

$$\varepsilon_{\mathrm{fit}} = \sum_{x} (\mu_{\mathrm{m}} - \mu_{\mathrm{f}})^2 = \sum_{x} \left(\mu_{\mathrm{m}} + \frac{E_t}{E_{x,y}}\right)^2 \tag{19.21}$$

© 2005 by Chapman & Hall/CRC

Differentiating with respect to the eight free parameters and setting the result to zero yields the linear solution

$$\left(\sum \phi\phi^{\mathrm{T}}\right)[a_{11},\, a_{12},\, b_1,\, a_{21},\, a_{22},\, b_2,\, c_1,\, c_2]^{\mathrm{T}} = \sum \left(x^{\mathrm{T}} E_{x,y} - E_t\right)\phi \tag{19.22}$$

where $\phi^{\mathrm{T}} = [E_x(x, y, 1), E_y(x, y, 1), xE_t - x^2E_x - xyE_y, xE_y - xyE_x - y^2E_y]$. This provides the approximate model of the transformation of the image h to the image g.

In order to compute the effectiveness of the transformation, the image h can be transformed to the image g using the parameters yielded by the above process. Then, some method of determining the difference between the two must be used. In order to yield the exact parameters of the transformation given the approximate model, Mann and Picard [37] utilized a "four-point method":

1. Select the four corners of the bounding box containing the region to be aligned. When the entire image is to be aligned, the four corners are the corners of the image. These points are denoted as $\mathbf{s} = [s_1,\, s_2,\, s_3,\, s_4]$.
2. Transform the region using the approximate model described above to yield the set of transformed corners $\mathbf{t} = \mathbf{u_m}(\mathbf{s})$.
3. The correspondences between $\mathbf{t}$ and $\mathbf{s}$ are given by solving the four linear equations

$$\begin{bmatrix} x'_k \\ y'_k \end{bmatrix} = \begin{bmatrix} x_k, y_k, 1, 0, 0, 0, -x_k x'_k, -y_k x'_k \\ 0, 0, 0, x_k, y_k, 1, -x_k y'_k, -y_k y'_k \end{bmatrix} \cdot \left[\, a_{x'x}, a_{x'y}, b_{x'}, a_{y'x}, a_{y'y}, b_{y'}, c_x, c_y \,\right]^{\mathrm{T}} \tag{19.23}$$

   for $1 < k < 4$ to give the parameters of the transformation $\mathbf{p}$.

The "four–point method" is applied repetitively to the image h until $\mathbf{p}$ represents the exact transformation parameters using the law of composition.

1. Set $h_0 = h$ and $\mathbf{p}_{0,0}$ to be the identity operator.
2. Repeat until the error between h_k and g falls below a certain threshold or a maximum number of iterations is reached:
   a. Estimate the eight parameters $\mathbf{q}_k$ of the approximate model between g and h_{k-1}.
   b. Relate $\mathbf{q}_k$ to the exact parameters using the "four-point method" to yield the new set of parameters $\mathbf{p}_k$.
   c. Apply the law of composition to accumulate the effect of the $\mathbf{p}_k$. The composite parameters are denoted $\mathbf{p}_{0,k}$ where $\mathbf{p}_{0,k} = \mathbf{p}_k \circ \mathbf{p}_{0,k-1}$. Finally, set $h_k = \mathbf{p}_{0,k} \circ \boldsymbol{h}$.

The CAMSHIFT (continuously adaptive mean shift) algorithm is designed to track human faces based on the mean shift algorithm [38]. In standard form, the mean shift algorithm attempts to find an object by finding the peak projection probability distribution image. The mean shift algorithm assumes that this probability distribution stays relatively constant as the object moves through an image [39]. CAMSHIFT modifies the mean shift algorithm by dropping this constancy assumption. CAMSHIFT is able to track dynamically changing probability distributions by using color histograms that are relatively insensitive to rotation and translation of an object.

Given a set of objects o and a local measure set M, for a single measurement vector m_k, at a point in the image the probability of one of the objects o_n containing the point is

$$p(o_n|m_k) = \frac{p(m_k|o_n)p(o_n)}{\sum_i p(m_k|o_i)p(o_i)} \tag{19.24}$$

© 2005 by Chapman & Hall/CRC

Equation (19.24) gives the probability of an object at a point in the image given a region surrounding that point, where a probability of 0.5 indicates complete uncertainty. The probability at each point in the vicinity of the last known position of the object is calculated, forming an object probability image. CAMSHIFT is then applied to this object probability image to track in which direction the object moved by climbing the gradient of the probability distribution to find the nearest peak.

CAMSHIFT involves five primary steps:

1. Select a search window size and shape for the probability distribution.
2. Place the search window.
3. Compute the probability distribution image of a region centered at the search window center but slightly larger than the window itself.
   a. Find the zeroth moment

$$M_{00} = \sum_x \sum_y I(x, y) \tag{19.25}$$

   b. Find the first moment for x and y

$$M_{10} = \sum_x \sum_y xI(x, y); \;\; M_{01} = \sum_x \sum_y yI(x, y) \tag{19.26}$$

   c. Find the mean search window location

$$x_c = \frac{M_{10}}{M_{00}}; \quad y_c = \frac{M_{01}}{M_{00}} \tag{19.27}$$

   where $I(x, y)$ is the pixel probability at position (x, y) in the image.
4. Shift the window to (x_c, y_c) and repeat step 3 until convergence.
5. For the next frame in the video sequence, center the window at the location obtained in step 4 and set the size of the window based upon the zeroth moment found there.

The mean shift calculation in step 3 will tend to converge to the mode of the distribution. This causes the CAMSHIFT algorithm to track the mode of color objects moving in the video scene. The second moments of the distribution are used to determine its orientation. The second moments are

$$M_{20} = \sum_x \sum_y x^2 I(x, y); \quad M_{02} = \sum_x \sum_y y^2 I(x, y) \tag{19.28}$$

and the major axis of the object being tracked is

$$\theta = \frac{\arctan\left(2\frac{\left((M_{11}/M_{10}) - x_c y_c\right)}{\left((M_{20}/M_{00}) - x_c^2\right) - \left((M_{02}/M_{00}) - y_c^2\right)}\right)}{2} \tag{19.29}$$

The length and width of the probability distribution region are found by solving the following equations:

$$a = \frac{M_{20}}{M_{00}} - x_c^2 \tag{19.30}$$

$$b = 2\left(\frac{M_{11}}{M_{00}} - x_c y_c\right) \tag{19.31}$$

© 2005 by Chapman & Hall/CRC

and

$$c = \frac{M_{02}}{M_{00}} - y_c^2 \tag{19.32}$$

Then the length I and width w are

$$I = \sqrt{\frac{(a+c) + \sqrt{b^2 + (a-c)^2}}{2}} \tag{19.33}$$

$$w = \sqrt{\frac{(a+c) - \sqrt{b^2 + (a-c)^2}}{2}} \tag{19.34}$$

The size of the window used in the search is adapted according to the first zeroth moment of the probability distribution

$$s = 2\sqrt{\frac{M_{00}}{256}} \tag{19.35}$$

This equation converts the area found to a number of pixels; 256 is the number of color intensity values possible in an 8-bit color scheme. These three concepts provide the basis for the fast alignment algorithm.

Omni perspective cameras are well suited to navigation and surveillance. They are relatively sensitive to the difference between small rotation and small translation, and provide information about a wide field of view. However, current implementations of omni cameras invariably suffer from poor resolution in some regions of their images [33]. In nearly direct opposition, planar perspective cameras offer high resolution throughout their images, but they suffer from a limited field of view and have difficulty distinguishing small rotation from small translation. Some hunting spiders use the combination of a pair of high-resolution forward-looking eyes with an array of lower resolution eyes that provide view in a wide field. The advantages of this type of system are obvious. Navigation and gross surveillance can be accomplished through the omni capabilities of the system, while tasks such as range-finding, object recognition and object manipulation can be addressed by the planar perspective capabilities. Inherent in this type of system is the ability of the control system to integrate the information gathered by all the vision sensors at its disposal.

In order to integrate the information gathered by an omni camera and a planar camera, we need to determine how the two sets of information overlap spatially. When time is not a constraining factor, there are a number of methods available for alignment of omni images and planar images. The geometry used to unwrap omni images into sets of planar images or a panoramic is relatively straightforward. With a set of planar images or panoramic derived from an omni image, there are a number of alignment algorithms that can be used to transform one image to the coordinate system of another. However, these operations are computationally expensive and are not suitable for video processing.

We present a system that provides on-demand alignment of any pair of image frames while still processing the frames at video rate. The method takes advantage of the fact that a histogram of a planar image projected into an omni image should be equivalent to the histogram of a subset of the omni image itself. The proposed method modifies CAMSHIFT to track the projection of a set of planar images through a set of omni images. CAMSHIFT tracks a dynamic probability distribution through a

© 2005 by Chapman & Hall/CRC

video sequence. The probability distribution may be dynamic due to translation and rotation of an object moving through the scene. We provide CAMSHIFT with a dynamically changing probability distribution to track. The current probability distribution of the planar image is supplied to CAMSHIFT frame by frame. CAMSHIFT tracks the dynamic probability distribution, changing because of a changing planar view instead of a rotating and translating object. CAMSHIFT tracks the object at video rate

CAMSHIFT requires an initial search window. For the first pair of frames, the first omni-image will be unwrapped into a panoramic image. The first planar camera image will then be aligned with this panoramic image, and the four corners of the best-fit transformation provide the initial search window. With a search window in place, the task becomes one of tracking the probability distribution of the moving planar image through the omni image. The probability distribution of each planar image is provided to CAMSHIFT for processing each omni image. The geometry of the search window is altered to reflect the shape of the planar image projection into the omni image. Pixels closer to the center of the omni image are weighted more heavily when generating probability distributions within the omni image. The modified CAMSHIFT algorithm tracks the probability distribution of the planar image through the omni image, giving an estimate of the projection location. When an exact alignment is required between any set of frames, only a subset of the panoramic image need be considered, as dictated by the projection estimate.

We assume that the focal point of the planar camera is aligned with the center of projection of the omni camera. If the planar camera can only pan and tilt, then the vertical edges of any image taken from the planar camera will be aligned with radial lines of the omni image. All planar projections into the omni image will occupy a sector. With this in mind, the CAMSHIFT algorithm is modified to use a search window that is always in the shape of a sector. Traditionally, the CAMSHIFT algorithm search window varies in terms of its size, shape, location, and orientation. In this application, shape and orientation are constrained to be those of a sector of the omni image. The projection of the color histogram of a planar image to the color histogram of a sector of unknown size and location is a critical component of the CAMSHIFT adaptation. Pixels at the top of the planar image must be weighted more heavily in order to account for compression of the planar image at the bottom of its sector projection.

CAMSHIFT takes the last known location of the object and follows a probability gradient to the most likely location of the object in the next frame. When tracking the projection of one image into another, there is the advantage of knowing the color histogram being sought at each frame. By deriving a method for transforming a planar color histogram into a sector color histogram, CAMSHIFT can be provided with an accurate description of what it is looking for frame-to-frame.

The function for the projection of the planar image into the omni image is dependent on the mirror in the catadioptric system. The omni camera used utilizes a paraboloid single mirror. Ideally, any algorithm for use with an omni camera will not require calibration for the shape of mirror used in any particular system. The nature of CAMSHIFT provides an opportunity to generalize to this effect. CAMSHIFT is robust to dynamically changing probability distributions, which in effect means that it tolerates shifts in the probability distribution it is tracking from one frame to the next. As CAMSHIFT will be provided with a true calculation of the probability distribution at each frame, it may be possible to generalize the plane-to-sector transformation in order to eliminate specific mirror equations from calculations.

Given a sector representing the projection of a planar image into the omni image, see Figure 19.22, it is possible to assume that the aspect ratio R of the planar image is reflected in the ratio of the sector height h to the length of the arc passing through x, y and bounded by the radial sides of the sector w' (Figure 19.23).

If the aspect ratio of the sector is constrained by the aspect ratio of the planar image, then it is possible to determine the exact dimensions of a sector given the coordinates of the sector center and area of the sector. If this assumption is not made, then an interesting problem arises. During fast alignment, the histogram of the planar image must be transformed to approximate the histogram of a corresponding sector in the omni image. For this to be done, either the exact parameters of the planar

© 2005 by Chapman & Hall/CRC

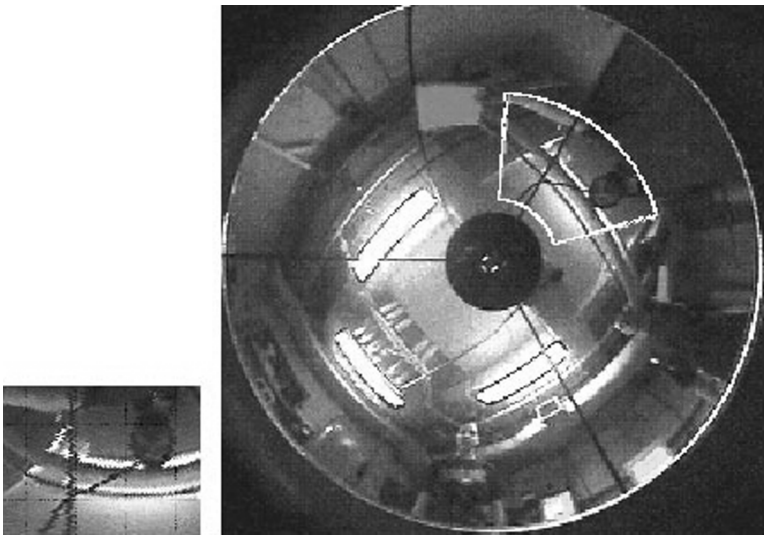


Figure 19.22. Projection of planar image into omni image.

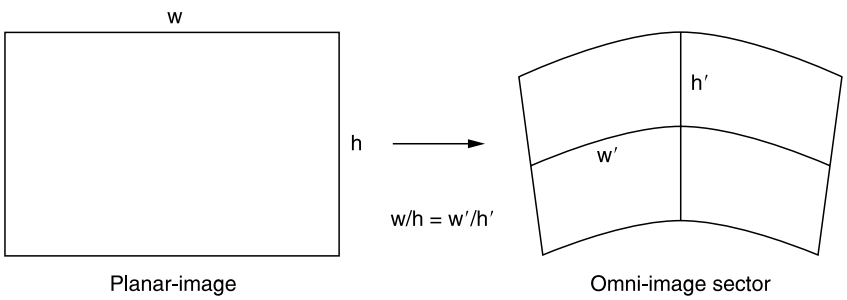


Figure 19.23. Projection ratio.

image projection into the omni image must be known or an assumption must be made about these parameters. Exact parameters can be obtained if the mirror equation is known and the exact location and area of the projection are known.

The mirror equation is available *a priori*, but the exact location and area of the projection are certainly not known, and are exactly what the algorithm attempts to estimate. Each estimation of position and area in the fast alignment algorithm depends on a past position and area of projection that is itself an estimate. Thus, the exact parameters are unavailable and an assumption must be made.

We expect the length of the median arc of the projection sector to be the most stable with respect to the height of the sector. As the sector area changes and the distance between the projection center and image center changes, so the outer and inner sector arc lengths will fluctuate relative to the sector height. For this reason, the median arc is chosen for the assumption needed. Measurements of the median arc length and the sector height show that the ratio between the two remains highly stable, fluctuating between 76 and 79%. Thus, the assumption that the aspect ratio of the virtual planar image is similar to the ratio of the sector median arc width to the sector height will be used in this thesis (Figure 19.24).

The distance from the center of the omni-image to the center of the sector is $r = \sqrt{(x^2 + y^2)}$. If the inner arc-edge of the sector is w_1 and the outer arc-edge is w_2, then the area of the ring bounded by

© 2005 by Chapman & Hall/CRC

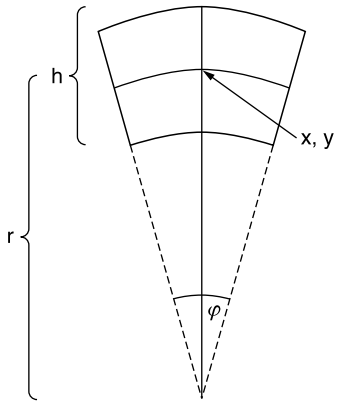


Figure 19.24. Sector centered at *x*, *y*.

circles centered at the image center and of radius r_1 and r_2, where $r_1 = r - h/2$ is the distance from image center to w_1 and $r_2 = r + h/2$ is the distance from image center to w_2, is given as

$$\pi(r + h/2)^2 - \pi(r - h/2)^2 = 2\pi rh \tag{19.36}$$

The area of the sector bounded by ϕ, w_1 and w_2 is given by

$$\frac{\text{AREA}}{2\pi rh} = \frac{\phi}{2\pi} \quad \text{AREA} = \phi rh \tag{19.37}$$

ϕ can be described in terms of r and w:

$$\frac{\phi}{2\pi} = \frac{w}{2\pi r} \tag{19.38}$$

Substituting $h = w/R$, where R is the aspect ratio of the planar image, into Equation (19.27) gives

$$\phi = \frac{Rh}{r} \tag{19.39}$$

Substituting Equation (19.39) into Equation (19.37) gives

$$\text{AREA} = Rh^2 \quad h = \sqrt{\frac{\text{AREA}}{R}} \quad \text{and} \quad w = R\sqrt{\frac{\text{AREA}}{R}} \tag{19.40}$$

This result is important for defining how the modified CAMSHIFT filter establishes a probability gradient and how the size of the actual search window itself is defined. When the initial alignment has been completed, CAMSHIFT establishes a probability gradient by moving the search window through the region directly surrounding the initial location. In a planar image application of CAMSHIFT, the search window is moved by translation in the *x*- and *y*- directions. In an omni image, the search window sector must be translated along the radius of the circle and the arcs of concentric circles within the omni image. With the results shown in Equations (39) and (40), the sector dimensions can be easily calculated as the center of the sector translates while searching, given the projection area. The area of the search window for the *n*th frame is a parameter returned by CAMSHIFT after the alignment of frame

© 2005 by Chapman & Hall/CRC

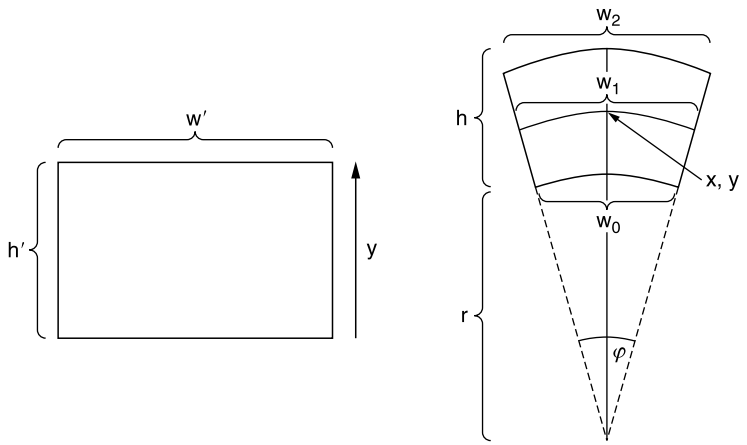


Figure 19.25. Diagram of histogram transformation geometry.

$n-1$. Given the dimensions of the search sector, it is possible to transform the color histogram of the planar image to fit the projection into the omni image.

If the arc length of w_0 is l_0, then weighting the pixels on the planar image can be done as follows. If the weight of a pixel in row $y=0$ (the bottom row) in the planar image is weight $= l_0/\mathrm{w}'$, then the weight of a pixel at row $y=n$ in the planar image is given as

$$\frac{r+nh/h'}{r} \times \text{weight} \tag{19.41}$$

This equation can easily transform the color histogram calculation of the planar image to fit the color histogram found in a corresponding sector. Figure 19.25 shows the histogram of a projection sector compared with transformed and untransformed planar image histograms. Clearly, the transformed planar histogram approximates the histogram of the original sector much more closely than does the untransformed planar histogram. The mean-square error (MSE) between the sector histogram and the warped planar histogram was less than 15% of the MSE between the sector histogram and the unwarped planar histogram.

19.8 Summary

This chapter provides an overview of data registration. Examples given here concentrated on two-dimensional images, since they are most intuitively understood. The information given here can easily be generalized to deal with data of an arbitrary number of dimensions. Registration is the process of finding the set of parameters that describe an accurate mapping of one set of data onto another set of data. The basic mathematical tools for affine mappings were described, and a survey of existing work was presented.

Of special interest is the ability to register data sets containing noise. To that end, we derived a number of objective functions that can be used to compare data registrations when the data have been corrupted. A number of different types of noise were considered. These objective functions are appropriate for registering data using optimization techniques [2,30].

We then presented two case studies for data registration using different sensing modalities. One approach used wavelet filters to register range images hierarchically. The other approach considered the geometries of different camera lenses, and explained how to maintain registration for a specific camera configuration in real time.

© 2005 by Chapman & Hall/CRC

References

[1] Gonzalez, R.C. and Woods, R.E., *Digital Image Processing*, Addison-Wesley, Menlo Park, 1992.

[2] Brooks, R.R. and Iyengar, S.S., *Multi-Sensor Fusion: Fundamentals and Applications with Software*, Prentice Hall, Upper Saddle River, NJ, 1998.

[3] Hill, F.S., *Computer Graphics*, Prentice Hall, Englewood, NJ, 1990.

[4] Wolberg, G., *Digital Image Warping*, IEEE, 1990.

[5] Faugeras, O., *Three-Dimensional Computer Vision a Geometric Viewpoint*, MIT Press, Cambridge, MA, 1993.

[6] Brown, L.G., A survey of image registration techniques, *ACM Computing Surveys*, 24(4), 1992, 325.

[7] Oghabian, M.A. and Todd-Pokropek, A., Registration of brain images by a multi-resolution sequential method, in *Information Processing in Medical Imaging*, Springer, New York, 1991, 165.

[8] Mandara, V.R. and Fitzpatrick, J.M., Adaptive search space scaling in digital image registration, *IEEE Transactions on Medical Imaging*, 8(3), 1989, 251.

[9] Palazzini, C.A. *et al.*, Interactive 3D patient-image registration, in *Information Processing in Medical Imaging*, Springer, New York, 1991, 132.

[10] Stockman, G. *et al.*, Matching images to models for registration and object detection via clustering, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 4(3), 1982, 229.

[11] Van Wie, P. and Stein, M., A LANDSAT digital image rectification system, *IEEE Transactions on Geoscience Electronics*, GE-15, 1977.

[12] Jain, J.R. and Jain, A.K., Displacement measurement and its application in interface image coding, *IEEE Transactions on Communications*, COM-29(12), 1981, 1799.

[13] Goshtasby, A., Piecewise linear mapping functions for image registration, *Pattern Recognition*, 19(6), 1986, 459.

[14] Wong, R.Y. and Hall, E.L., Performance comparison of scene matching techniques, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(3), 1979, 325.

[15] Mitiche, A. and Aggarwal, K., Contour registration by shape specific points for shape matching comparison, *Vision, Graphics and Image Processing*, 22, 1983, 396.

[16] Barnea, D. and Silverman, H., A class of algorithms for fast digital image registration, *IEEE Transactions on Computers*, C-21 (2), 1972, 179.

[17] Horn, B.R. and Bachman, B.L., Using synthetic images with surface models, *Communications of the ACM*, 21, 1977, 914.

[18] Barrow, H.G. *et al.*, Parametric correspondence and chamfer matching: two new techniques for image matching, in *Proceedings of International Joint Conference on Artificial Intelligence*, 1977, 659.

[19] Pinz, A. *et al.*, Affine matching of intermediate symbolic presentations, in *CAIP '95 Proceedings LNCS 970*, Hlavac and Sara, (eds), Springer Verlag, Berlin, 1995, 132.

[20] Chellappa, R. *et al.*, On the positioning of multisensory imagery for exploitation and target recognition, *Proceedings of the IEEE*, 85(1), 1997, 120.

[21] Zhou, Y. *et al.*, Registration of mobile sensors using the parallelized extended Kalman filter, *Optical Engineering*, 36(3), 1997, 780.

[22] Grewe, L. and Brooks, R.R., Efficient registration in the compressed domain, *Wavelet Applications VI*, *SPIE Proceedings*, Vol. 3723, Szu, H. (ed.), Aerosense 1999, Orlando, FL.

[23] Sharman, R. *et al.*, Wavelet based registration and compression of sets of images, *SPIE Proceedings* 3078, 497.

[24] DeVore, R.A. *et al.*, Using nonlinear wavelet compression to enhance image registration, *SPIE Proceedings*, 3078, 539.

[25] Barhen, J. and Protopopescu, V., Generalized TRUST algorithms for global optimization, *State of the Art in Global Optimization*, Floudas, C.A. and Pardalos, P.M. (eds), Kluwer Academic Publishers, 1996, 163.

© 2005 by Chapman & Hall/CRC

[26] Barhen, J. *et al.*, TRUST: a deterministic algorithm for global optimization, *Science*, 276, 16 May, 1997.
[27] Cetin, B.C. *et al.*, Teminal repeller unconstrained subenergy tunneling (TRUST) for fast global optimization, *Journal of Optimization Theory and Application*s, 77(l), 1993, 97.
[28] Brooks, R.R. *et al.*, Automatic correlation and calibration of noisy sensor readings using elite genetic algorithms, *Artificial Intelligence*, 84, 1996, 339.
[29] Brooks, R.R., Robust sensor fusion algorithms: calibration and cost minimization, Ph.D. dissertation, Louisiana State University, August 1996.
[30] Chen, Y. *et al.*, Efficient global optimization for image registration, *IEEE Transactions on Knowledge and Data Engineering*, 14(1), 2002, 79, http://www.computer.org/tkde/Image-processing.html.
[31] Russ, J.C., *The Image Processing Handbook*, CRC Press, Boca Raton, FL, 1995.
[32] Grewe, L. and Brooks, R., On localization of objects in the wavelet domain, in *1997 IEEE Symposium on Computational Intelligence in Robotics and Automation*, July 1997, 412.
[33] Nayar, S., Catadioptric omnidirectional camera, in *Computer Vision and Pattern Recognition, 1997. Proceedings, 1997 IEEE Computer Society Conference*, 1997, 482.
[34] Rees, D., Panoramic televison viewing system, U.S. Patent, 3,505,465, April, 1970.
[35] Daniilidis, K. and Geyer, C., Omnidirectional vision: theory and algorithms, in *Pattern Recognition, 2000. Proceedings. 15th International Conference*, 2000, vol. 1, 89.
[36] Nayar, S. and Peri, V., Folded catadioptric cameras, in *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference*, 1999, vol. 2, 223.
[37] Mann, S. and Picard, R., Video orbits of the projective group a simple approach to featureless estimation of parameters, *IEEE Transactions on Image Processing*, 6(9), 1997, 1281.
[38] Bradski, G., Real time face and object tracking as a component of a perceptual user interface, in *Applications of Computer Vision, 1998. WACV '98. Proceedings*, *Fourth IEEE Workshop*, 1998, 214.
[39] Cheng, Y., Mean shift, mode seeking, and clustering, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17, 1998, 790.

© 2005 by Chapman & Hall/CRC

20

Signal Calibration, Estimation for Real-Time Monitoring and Control*

Asok Ray and Shashi Phoha

20.1 Introduction

Performance, reliability, and safety of complex dynamical processes such as aircraft and power plants depend upon validity and accuracy of sensor signals that measure plant conditions for information display, health monitoring, and control [1]. Redundant sensors are often installed to generate spatially averaged time-dependent estimates of critical variables so that reliable monitoring and control of the plant are assured. Examples of redundant sensor installations in complex engineering applications are:

- Inertial navigational sensors in both tactical and transport aircraft for guidance and control [2,3].
- Neutron flux detectors in the core of a nuclear reactor for fuel management, health monitoring, and power control [4].
- Temperature, pressure, and flow sensors in both fossil-fuel and nuclear steam power plants for health monitoring and feedforward–feedback control [5].

Sensor redundancy is often augmented with analytical measurements that are obtained from physical characteristics and/or model of the plant dynamics in combination with other available sensor data [4,6]. The redundant sensors and analytical measurements are referred to as redundant measurements in the following.

Individual measurements in a redundant set may often exhibit deviations from each other after a length of time. These differences could be caused by slowly time-varying sensor parameters (e.g. amplifier gain), plant parameters (e.g. structural stiffness and heat transfer coefficient), transport delays, etc. Consequently, some of the redundant measurements could be deleted by a fault detection and

*The research work reported in this chapter has been supported in part by the Army Research Office under Grant No. DAAD19-01-1-0646.

© 2005 by Chapman & Hall/CRC

isolation (FDI) algorithm [7] if they are not calibrated periodically. On the other hand, failure to isolate a degraded measurement could cause an inaccurate estimate of the measured variable by, for example, increasing the threshold bound in the FDI algorithm. In this case, the plant performance may be adversely affected if that estimate is used as an input to the decision and control system. This problem can be resolved by adaptively filtering the set of redundant measurements as follows:

- All measurements, which are consistent relative to the threshold of the FDI algorithm, are simultaneously calibrated on-line to compensate for their relative errors.
- The weights of individual measurements for computation of the estimate are adaptively updated on-line based on their respective *a posteriori* probabilities of failure instead of being fixed *a priori*.

In the event of an abrupt disruption of a redundant measurement in excess of its allowable bound, the respective measurement is isolated by the FDI logic, and only the remaining measurements are calibrated to provide an unbiased estimate of the measured variable. On the other hand, if a gradual degradation (e.g. a sensor drift) occurs, the faulty measurement is not immediately isolated by the FDI logic. However, its influence on the estimate and calibration of the remaining measurements is diminished as a function of the magnitude of its residual (i.e. deviation from the estimate), which is an indicator of its degradation. This is achieved by decreasing the relative weight of the degraded measurement as a monotonic function of its deviation from the remaining measurements. Thus, if the error bounds of the FDI algorithm are appropriately increased to reduce the probability of false alarms, then the resulting delay in detecting a gradual degradation could be tolerated. The rationale is that an undetected fault, as a result of the adaptively reduced weight, would have smaller bearing on the accuracy of measurement calibration and estimation. Furthermore, since the weight of a gradually degrading measurement is smoothly reduced, the eventual isolation of the fault would not cause any abrupt change in the estimate. This feature, known as *bumpless* transfer in the process control literature, is very desirable for plant operation.

This chapter presents a calibration and estimation filter for redundancy management of sensor data and analytical measurements. The filter is validated based on redundant sensor data of throttle steam temperature collected from an operating power plant. Development and validation of the filter algorithm are presented in the main body of the chapter along with concluding remarks. Appendix A presents the theory of multiple hypotheses based on the *a posteriori* probability of failure of a single measurement [8].

20.2 Signal Calibration and Measurement Estimation

A redundant set of ℓ sensors and/or analytical measurements of an n-dimensional plant variable are modeled at the kth sample as

$$m_k = (H + \Delta H_k)x_k + b_k + e_k \tag{20.1}$$

where m_k is the $(\ell \times 1)$ vector of (uncalibrated) redundant measurements; H is the $(\ell \times n)$ *a priori* determined matrix of scale factor having rank n, with $\ell > n \geq 1$; ΔH_k is the $(\ell \times n)$ matrix of scale factor errors; x_k is the $(n \times 1)$ vector of true (unknown) value of the measured variable; b_k is the $(\ell \times 1)$ vector of bias errors; and e_k is the $(\ell \times 1)$ vector of measurement noise, such that $E[e_k] = 0$ and $E[e_k e_l^{\mathrm{T}}] = R_k \delta_{kl}$.

The noise covariance matrix R_k of uncalibrated measurements plays an important role in the adaptive filter for both signal calibration and measurement estimation. It is shown in the following how R_k is recursively tuned based on the history of calibrated measurements.

Equation (20.1) is rewritten in a more compact form as

$$m_k = Hx_k + c_k + e_k \tag{20.2}$$

© 2005 by Chapman & Hall/CRC

where the correction c_k due to the combined effect of bias and scale-factor errors is defined as

$$c_k \equiv \Delta H_k x_k + b_k \tag{20.3}$$

The objective is to obtain an unbiased predictor estimate $\hat{c}_k$ of the correction c_k so that the sensor output m_k can be calibrated at each sample. A recursive relation of the correction c_k is modeled similar to a random walk process as

$$\begin{aligned} c_{k+1} &= c_k + v_k \\ E[v_k] &= 0; \quad E[v_k v_j] = Q\delta_{kj} \text{ and } E[v_k e_j] = 0 \quad \forall k, j \end{aligned} \tag{20.4}$$

where the stationary noise v_k represents uncertainties of the model in Equation (20.4).

We construct a filter to calibrate each measurement with respect to the remaining redundant measurements. The filter input is the parity vector p_k of the uncalibrated measurement vector m_k, which is defined [2,9] as

$$p_k = Vm_k \tag{20.5}$$

where the rows of the projection matrix $V \in \Re^{(\ell-n)\times\ell}$ form an orthonormal basis of the left null space of the measurement matrix $H \in \Re^{\ell\times n}$ in Equation (20.1), i.e.

$$\begin{aligned} VH &= 0_{(\ell-n)\times n} \\ VV^{\mathrm{T}} &= I_{(\ell-n)\times(\ell-n)} \end{aligned} \tag{20.6}$$

and the columns of V span the parity space that contains the parity vector. A combination of Equations (20.2), (20.4), (20.5) and (20.6) yields

$$p_k = Vc_k + \varepsilon_k \tag{20.7}$$

where the noise $\varepsilon_k \equiv Ve_k$ having $E[\varepsilon_k] = 0$ and $E[\varepsilon_k \varepsilon_j^{\mathrm{T}}] \equiv VR_kV^{\mathrm{T}}\delta_{kj}$. If the scale-factor error matrix ΔH_k belongs to the column space of H, then the parity vector p_k is independent of the true value x_k of the measured variable. Therefore, for $\|V\Delta H_k x_k\| \ll \|Vb_k\|$, which includes relatively small scale-factor errors, the calibration filter operates approximately independent of x_k.

Now we proceed to construct a recursive algorithm to predict the estimated correction $\hat{c}_k$ based on the principle of best linear least-squares estimation that has the structure of an optimal minimum-variance filter [10,11] and uses Equations (20.4) and (20.7):

$$\left.\begin{aligned} \hat{c}_{k+1} &= \hat{c}_k + K_k\gamma_k && \text{given } \hat{c}_0 \\ P_{k+1} &= (I - K_kV)P_k + Q && \text{given } P_0 \text{ and } Q \\ K_k &= P_kV^{\mathrm{T}}\left(V[R_k + P_k]V^{\mathrm{T}}\right)^{-1} && \text{given } R_k \\ \gamma_k &= p_k - V\hat{c}_k && \text{innovation} \end{aligned}\right\} \tag{20.8}$$

Upon evaluation of the unbiased estimated correction $\hat{c}_k$, the uncalibrated measurement m_k is compensated to yield the calibrated measurement y_k as

$$y_k = m_k - \hat{c}_k \tag{20.9}$$

© 2005 by Chapman & Hall/CRC

Using Equations (20.5) and (20.9), the innovation γ_k in Equation (20.8) can be expressed as the projection of the calibrated measurement y_k onto the parity space, i.e.

$$\gamma_k = V y_k \tag{20.10}$$

By setting $\Gamma_k \equiv K_k V$, we obtain an alternative form of the recursive relations in Equation (20.8) as

$$\begin{aligned} \hat{c}_{k+1} &= \hat{c}_k + \Gamma_k y_k && \text{given } \hat{c}_0 \\ P_{k+1} &= (I - \Gamma_k) P_k + Q && \text{given } P_0 \text{ and } Q \\ \Gamma_k &= P_k V^{\mathrm{T}} \left(V[R_k + P_k] V^{\mathrm{T}} \right)^{-1} V && \text{given } R_k \end{aligned} \tag{20.11}$$

Note that the inverse of the matrix $\left(V[R_k + P_k]V^{\mathrm{T}}\right)$ in Equations (20.8) and (20.11) exists because the rows of V are linearly independent, $R_k > 0$, and $P_k \geq 0$.

Next we obtain an unbiased weighted least squares estimate $\hat{x}_k$ of the measured variable x_k based on the calibrated measurement y_k as

$$\hat{x}_k = (H^{\mathrm{T}} R_k^{-1} H)^{-1} H^{\mathrm{T}} R_k^{-1} y_k \tag{20.12}$$

The inverse of the (symmetric positive-definite) measurement covariance matrix R_k serves as the weighting matrix for generating the estimate $\hat{x}_k$, and is used as a filter matrix. Compensation of a (slowly varying) undetected error in the jth measurement out of ℓ redundant measurements causes the largest jth element $\left|{}_j\hat{c}_k\right|$ in the correction vector $\hat{c}_k$. Therefore, a limit check on the magnitude of each element of $\hat{c}_k$ will allow detection and isolation of the degraded measurement. The bounds of limit check, which could be different for the individual elements of $\hat{c}_k$, are selected by trade-off between the probability of false alarms and the allowable error in the estimate $\hat{x}_k$ of the measured variable [12].

20.2.1 Degradation Monitoring

Following Equation (20.12), we define the residual η_k of the calibrated measurement y_k as

$$\eta_k = y_k - H\hat{x}_k \tag{20.13}$$

The residuals represent a measure of relative degradation of individual measurements. For example, under the normal condition, all calibrated measurements are clustered together, i.e. $\|\eta_k\| \approx 0$, although this may not be true for the residual $(m_k - H\hat{x}_k)$ of uncalibrated measurements.

While large abrupt changes in excess of the error threshold are easily detected and isolated by a standard diagnostics procedure (e.g. [7]), small errors (e.g. slow drift) can be identified from the *a posteriori* probability of failure that is recursively computed from the history of residuals based on the following trinary hypotheses:

$$\begin{aligned} H^0 &: \textit{Normal behavior with} \text{ a priori } \textit{conditional density function } {}_jf^0(\cdot) \equiv {}_jf\left(\cdot \middle| H^0\right) \\ H^1 &: \textit{High (positive) failure with} \text{ a priori } \textit{conditional density function } {}_jf^1(\cdot) \equiv {}_jf\left(\cdot \middle| H^1\right) \\ H^2 &: \textit{Low (negative) failure with} \text{ a priori } \textit{conditional density function } {}_jf^2(\cdot) \equiv {}_jf\left(\cdot \middle| H^2\right) \end{aligned} \tag{20.14}$$

where the left subscript refers to of the jth measurement for $j = 1, 2, \ldots, \ell$, and the right superscript indicates the normal behavior or failure mode. The density function for each residual is determined

© 2005 by Chapman & Hall/CRC

a priori from experimental data and/or instrument manufacturers' specifications. Only one test is needed here to accommodate both positive and negative failures; this is in contrast to the binary hypotheses, which require two tests. We now apply the recursive relations for multi-level hypotheses testing of single variables, derived in Appendix A, to each residual of the redundant measurements. Then, for the jth measurement at the kth sampling instant, *a posteriori* probability of failure ${}_j\Pi_k$ is obtained following Equation (20.A17) as

$$\left.\begin{aligned} {}_j\Psi_k &= \left(\frac{{}_jp + {}_j\Psi_{k-1}}{2(1-{}_jp)}\right)\left(\frac{{}_jf^1({}_j\eta_k) + {}_jf^2({}_j\eta_k)}{{}_jf^0({}_j\eta_k)}\right) \\ {}_j\Pi_k &= \frac{{}_j\Psi_k}{1+{}_j\Psi_k} \end{aligned}\right\} \tag{20.15}$$

where ${}_jp$ is the *a priori* probability of failure of the jth sensor during one sampling period, and the initial condition of each state, ${}_j\Psi_0, j = 1, 2, \ldots, \ell$, needs to be specified.

Based on the *a posteriori* probability of failure, we now proceed to formulate a recursive relation for the measurement noise covariance matrix R_k that influences both calibration and estimation as seen in Equations (20.8) to (20.12). Its initial value R_0, which is determined *a priori* from experimental data and/or instrument manufacturers' specifications, provides the *a priori* information on individual measurement channels and conforms to the normal operating conditions when all measurements are clustered together, i.e. $\|\eta_k\| \approx 0$. In the absence of any measurement degradation, R_k remains close to its initial value R_0. Significant changes in R_k may take place if one or more sensors start degrading. This phenomenon is captured by the following model:

$$R_k = \sqrt{R_k^{\text{rel}}}R_0\sqrt{R_k^{\text{rel}}} \quad \text{with} \quad R_0^{\text{rel}} = I \tag{20.16}$$

where R_k^{rel} is a positive-definite diagonal matrix representing relative performance of the individual calibrated measurements and is recursively generated as follows:

$$R_{k+1}^{\text{rel}} = \text{diag}\big[h({}_j\Pi_k)\big], \quad \text{i.e. } {}_jr_{k+1}^{\text{rel}} = h({}_j\Pi_k) \tag{20.17}$$

where ${}_jr_k^{\text{rel}}$ and ${}_j\Pi_k$ are respectively the relative variance and *a posteriori* probability of failure of the jth measurement at the kth instant; and h: $[0, 1) \to [1, \infty)$ is a continuous monotonically increasing function with boundary conditions $h(0) = 1$ and $h(\varphi) \to \infty$ as $\varphi \to 1$.

The implication of Equation (20.17) is that the credibility of a sensor monotonically decreases with increase in its variance, which tends to infinity as its *a posteriori* probability of failure approaches unity. The magnitude of the relative variance ${}_jr_k^{\text{rel}}$ is set to the minimum value of one for zero *a posteriori* probability of failure. In other words, the jth diagonal element ${}_jw_k^{\text{rel}} \equiv 1/{}_jr_k^{\text{rel}}$ of the weighting matrix $W_k^{\text{rel}} \equiv (R_k^{\text{rel}})^{-1}$ tends to zero as ${}_j\Pi_k$ approaches unity. Similarly, the relative weight ${}_jw_k^{\text{rel}}$ is set to the maximum value of one for ${}_j\Pi_k = 0$. Consequently, a gradually degrading sensor carries monotonically decreasing weight in the computation of the estimate $\hat{\boldsymbol{x}}_k$ in Equation (20.12).

Next we set the bounds on the states ${}_j\Psi_k$ of the recursive relation in Equation (20.15). The lower limit of ${}_j\Pi_k$ (which is an algebraic function of ${}_j\Psi_k$) is set to the probability ${}_jp$ of intra-sample failure. On the other extreme, if ${}_j\Pi_k$ approaches unity, then the weight ${}_jw_k^{\text{rel}}$ (which approaches zero) may prevent fast restoration of a degraded sensor following its recovery. Therefore, the upper limit of ${}_j\Pi_k$ is set to $(1 - {}_j\alpha)$, where ${}_j\alpha$ is the allowable probability of false alarms of the jth measurement. Consequently, the function $h(\cdot)$ in Equation (20.17) is restricted to the domain $[{}_jp, (1 - {}_j\alpha)]$ to account for probabilities of intra-sampling failures and false alarms. Following Equation (20.15), the lower and

© 2005 by Chapman & Hall/CRC

upper limits of the states ${}_j\Psi_k$ are thus become ${}_jp/(1-{}_jp)$ and $(1-{}_j\alpha)/{}_j\alpha$ respectively. Consequently, the initial state in Equation (20.15) is set as ${}_j\Psi_0 = {}_jp/(1-{}_jp)$ for $j = 1, 2, \ldots, \ell$.

20.2.2 Possible Modifications of the Calibration Filter

The calibration filter is designed to operate in conjunction with an FDI system that is capable of detecting and isolating abrupt disruptions (in excess of specified bounds) in one or more of the redundant measurements [7]. The consistent measurements, identified by the FDI system, are simultaneously calibrated at each sample. Therefore, if a continuous degradation, such as a gradual monotonic drift of a sensor amplifier, occurs sufficiently slowly relative to the filter dynamics, then the remaining (healthy) measurements might be affected, albeit by a small amount, due to simultaneous calibration of all measurements, including the degraded measurement. Thus, the fault may be disguised in the sense that a very gradual degradation over a long period may potentially cause the estimate $\hat{\boldsymbol{x}}_k$ to drift. This problem could be resolved by modifying the calibration filter with one or both of the following procedures:

- *Adjustments via limit check on the correction vector* $\hat{\boldsymbol{c}}_k$. Compensation of a (slowly varying) undetected error in the jth measurement out of ℓ redundant measurements will cause the largest jth element $|{}_j\hat{c}_k|$ in the correction vector $\hat{\boldsymbol{c}}_k$. Therefore, a limit check on the magnitude of each element of $\hat{\boldsymbol{c}}_k$ will allow detection and isolation of the degraded measurement. The bounds of limit check, which could be different for the individual elements of $\hat{\boldsymbol{c}}_k$, are selected by trade-off between the probability of false alarms and the allowable error in the estimate $\hat{\boldsymbol{x}}_k$ of the measured variable [12].
- *Usage of additional analytical measurements*. If the estimate $\hat{\boldsymbol{x}}_k$ is used to generate an analytic measurement of another plant variable that is directly measured by its own sensor(s), then a possible drift of the calibration filter can be detected whenever this analytical measurement disagrees with the sensor data in excess of a specified bound. The implication is that either the analytical measurement or the sensor is faulty. Upon detecting such a fault, the actual cause needs to be identified based on additional information, including reasonability check. This procedure not only checks the calibration filter but also guards against simultaneous and identical failure of several sensors in the redundant set, possibly due to a common cause, known as the common-mode fault.

20.3 Sensor Calibration in a Commercial-Scale Fossil-Fuel Power Plant

The calibration filter, derived above, has been validated in a 320 MWe coal-fired supercritical power plant for on-line sensor calibration and measurement estimation at the throttle steam condition of ~1040°F (560°C) and ~3625 psia (25.0 MPa). The set of redundant measurements is generated by four temperature sensors installed at different spatial locations of the main steam header that carries superheated steam from the steam generator into the high-pressure turbine via the throttle valves and governor valves [13]. Since these sensors are not spatially collocated, they can be asynchronous under transient conditions due to the transport lag. The filter simultaneously calibrates the sensors to generate a time-dependent estimate of the throttle steam temperature that is spatially averaged over the main steam header. This information on the estimated average temperature is used for health monitoring and damage prediction in the main steam header, as well as for coordinated feedforward–feedback control of the power plant under both steady-state and transient operations [14,15]. The filter software is hosted in a Pentium platform.

The readings of all four temperature sensors have been collected over a period of 100 h at a sampling frequency of once every 1 min. The data collected, after bad-data suppression (e.g. elimination of obvious outliers following built-in tests, such as limit check and rate check), show that each sensor

© 2005 by Chapman & Hall/CRC

exhibits temperature fluctuations resulting from the inherent thermal–hydraulic noise and process transients, as well as the instrumentation noise. For this specific application, the parameters, functions, and matrices of the calibration filter are selected as described below.

20.3.1 Filter Parameters and Functions

We start with the filter parameters and functions that are necessary for degradation monitoring. In this application, each element of the residual vector η_k of the calibrated measurement vector y_k is assumed to be Gaussian distributed that assures existence of the likelihood ratios in Equation (20.15). The structures of the *a priori* conditional density functions are chosen as follows:

$$
\begin{aligned}
{}_jf^0(\varphi) &= \frac{1}{\sqrt{2\pi}\,{}_j\sigma}\exp\left(-\tfrac{1}{2}\left(\frac{\varphi}{{}_j\sigma}\right)^2\right)\\
{}_jf^1(\varphi) &= \frac{1}{\sqrt{2\pi}\,{}_j\sigma}\exp\left(-\tfrac{1}{2}\left(\frac{\varphi-{}_j\theta}{{}_j\sigma}\right)^2\right)\\
{}_jf^2(\varphi) &= \frac{1}{\sqrt{2\pi}\,{}_j\sigma}\exp\left(-\tfrac{1}{2}\left(\frac{\varphi+{}_j\theta}{{}_j\sigma}\right)^2\right)
\end{aligned}
\tag{20.18}
$$

where ${}_j\sigma$ is the standard deviation, and ${}_j\theta$ and $-{}_j\theta$ are the thresholds for positive and negative failures respectively of the jth residual.

Since it is more convenient to work in the natural-log scale for Gaussian distribution than for the linear scale, an alternative to Equation (20.17) is to construct a monotonically decreasing continuous function g: $(-\infty, 0) \to (0, 1]$ in lieu of the monotonically increasing continuous function h: $[0, 1) \to [1, \infty)$, so that

$$
W_{k+1}^{\text{rel}} \equiv \left(R_{k+1}^{\text{rel}}\right)^{-1} = \text{diag}\left[g\left(\ln {}_j\Pi_k\right)\right], \text{ i.e. the weight } {}_jw_{k+1}^{\text{rel}} \equiv \left({}_jr_{k+1}^{\text{rel}}\right)^{-1} = g\left(\ln {}_j\Pi_k\right) \tag{20.19}
$$

The linear structure of the continuous function $g(\cdot)$ is chosen to be piecewise linear, as given below:

$$
g(\varphi) = \begin{cases} w^{\max} & \text{for } \varphi \le \varphi^{\min} \\ \dfrac{(\varphi^{\max}-\varphi)w^{\max} + (\varphi-\varphi^{\min})w^{\min}}{\varphi^{\max}-\varphi^{\min}} & \text{for } -\infty \le \varphi^{\min} \le \varphi \le \varphi^{\max} < 0 \\ w^{\min} & \text{for } \varphi \ge \varphi^{\max} \end{cases} \tag{20.20}
$$

The function $g(\cdot)$ maps the space of ${}_j\Pi_k$ in the log scale into the space of the relative weight ${}_jw_{k+1}^{\text{rel}}$ of individual sensor data. The domain of $g(\cdot)$ is restricted to $[\ln({}_jp), \ln(1-{}_j\alpha)]$ to account for probability ${}_jp$ of intra-sampling failure and probability ${}_j\alpha$ of false alarms for each of the four sensors. The range of $g(\cdot)$ is selected to be $[{}_jw^{\min}, 1]$ where a positive minimum weight (i.e. ${}_jw^{\min} > 0$) allows the filter to restore a degraded sensor following its recovery. Numerical values of the filter parameters ${}_j\sigma$, ${}_j\theta$, ${}_jp$, ${}_j\alpha$, and ${}_jw^{\min}$ are presented below:

- The standard deviations of the *a priori* Gaussian density functions of the four temperature sensors are:
  ${}_1\sigma = 4.1°\text{F}(2.28°\text{C})$; ${}_2\sigma = 3.0°\text{F}(1.67°\text{C})$; ${}_3\sigma = 2.4°\text{F}(1.33°\text{C})$; ${}_4\sigma = 2.8°\text{F}(1.56°\text{C})$

The initial condition for the measurement noise covariance matrix is set as $R_0 = \text{diag}[{}_j\sigma]$. The failure threshold parameters are selected as: ${}_j\theta = {}_j\sigma/2$ for $j = 1, 2, 3, 4$.

© 2005 by Chapman & Hall/CRC

- The probability of intra-sampling failure is assumed to be identical for all four sensors, as they are similar in construction and operate under an identical environment. Operation experience at the power plant shows that the mean life of a resistance thermometer sensor, installed on the main steam header, is about 700 days (i.e. about 2 years) of continuous operation. For a sampling interval of 1 min, this information leads to

  $${}_jp \approx 10^{-6} \quad \text{for } j = 1, 2, 3, 4$$

- The probability of false alarms is selected in consultation with the plant operating personnel. On average, each sensor is expected to generate a false alarm after approximately 700 days of continuous operation (i.e. once in 2 years). For a sampling interval of 1 min, this information leads to

  $${}_j\alpha \approx 10^{-6} \quad \text{for } j = 1, 2, 3, 4$$

- To allow restoration of a degraded sensor following its recovery, the minimum weight is set as

  $${}_jw_{\min} \approx 10^{-3} \quad \text{for } j = 1, 2, 3, 4$$

20.3.2 Filter Matrices

After conversion of the four temperature sensor data into engineering units, the scale factor matrix in Equation (1) becomes: $H = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}^{\mathrm{T}}$. Consequently, following Potter and Suman and Ray and Luck (1991), the parity space projection matrix in Equation (20.6) becomes

$$V = \begin{bmatrix} \sqrt{\frac{3}{4}} & -\sqrt{\frac{1}{12}} & -\sqrt{\frac{1}{12}} & -\sqrt{\frac{1}{12}} \\ 0 & \sqrt{\frac{2}{3}} & -\sqrt{\frac{1}{6}} & -\sqrt{\frac{1}{6}} \\ 0 & 0 & \sqrt{\frac{1}{2}} & -\sqrt{\frac{1}{2}} \end{bmatrix}$$

In the event of a sensor being isolated as faulty, sensor redundancy reduces to three, for which

$$H = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}^{\mathrm{T}} \quad \text{and} \quad V = \begin{bmatrix} \sqrt{\frac{2}{3}} & -\sqrt{\frac{1}{6}} & -\sqrt{\frac{1}{6}} \\ 0 & \sqrt{\frac{1}{2}} & -\sqrt{\frac{1}{2}} \end{bmatrix}$$

The ratio, $R_k^{-1/2} Q R_k^{-1/2}$, of covariance matrices Q and R_k largely determines the characteristics of the minimum variance filter in Equation (20.8) or Equation (20.11). The filter gain Γ_k increases with a larger ratio $R_k^{-1/2} Q R_k^{-1/2}$ and *vice versa*. Since the initial steady-state value R_0 is specified and R_k^{rel} is recursively generated thereon to calculate R_k via Equation (20.16), the choice is left only for selection of Q. As *a priori* information on Q may not be available, its choice relative to R_0 is a design feature. In this application, we have set $Q = R_0$.

20.3.3 Filter Performance Based on Experimental Data

The filter was tested on-line in the power plant over a continuous period of 9 months except for two short breaks during plant shutdown. The test results showed that the filter was able to calibrate each

© 2005 by Chapman & Hall/CRC

sensor under both pseudo-steady-state and transient conditions under closed-loop control of throttle steam temperature. The calibrated estimate of the throttle steam temperature was used for plant control under steady-state, load-following, start-up, and scheduled shutdown conditions. No natural failure of the sensors occurred during the test period, and there was no evidence of any drift of the estimated temperature. As such, the modifications (e.g. adjustments via limit check on $\hat{c}_k$, and additional analytical measurements) of the calibration filter, described earlier in this chapter, were not implemented. In addition to testing under on-line plant operation, simulated faults have been injected into the plant data to evaluate the efficacy of the calibration filter under sensor failure conditions. Based on the data of four temperature sensors that were collected at an interval of 1 min over a period of 0 to 100 h, the following three cases of simulated sensor degradation are presented below:

20.3.3.1 Case 1 (Drift Error and Recovery in a Single Sensor)

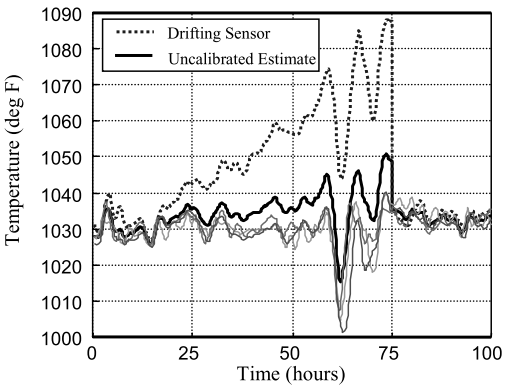
Starting at 12.5 h, a drift error was injected into the data stream of Sensor#1 in the form of an additive ramp at the rate of 1.167°F (0.648°C) per hour. The injected fault was brought to zero at 75 h, signifying that the faulty amplifier in the sensor hardware was corrected and reset.

The simulation results in Figure 20.1 exhibit how the calibration filter responds to a gradual drift in one of the four sensors while the remaining three are normally functioning. Figure 20.1(a) shows the response of the four uncalibrated sensors and the estimate generated by simple averaging (i.e. fixed identical weights) of these four sensor readings at each sample. The sensor data profile includes transients lasting from ~63 to ~68 h. From time 0 to 12.5 h, when no fault is injected, all sensor readings are clustered together. Therefore, the uncalibrated estimate, shown by a thick solid line, is in close agreement with all four sensors during the period 0 to 12.5 h. Sensor#1, shown by the dotted line, starts drifting at 12.5 h while the remaining sensors stay healthy. Consequently, the uncalibrated estimate starts drifting at one quarter of the drift rate of Sensor#1 because of equal weighting of all sensors in the absence of the calibration filter. Upon termination of the drift fault at 75 h, when Sensor#1 is brought back to the normal state, the uncalibrated estimate resumes its normal state close to all four sensors for the remaining period from 75 to 100 h.

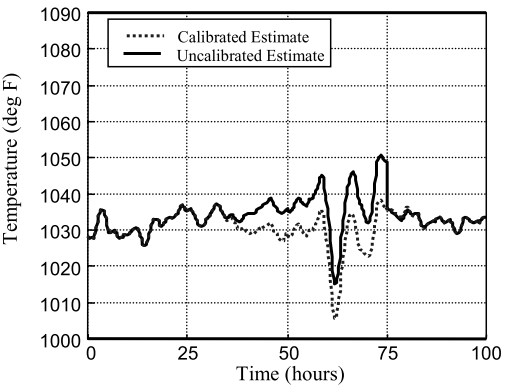
In Figure 20.1(b) shows the response of the four calibrated sensors and the estimate generated by weighted averaging (i.e. varying nonidentical weights) of these four sensor readings at each sample. The calibrated estimate in (b) stays with the remaining three healthy sensors even though Sensor#1 is gradually drifting. Figure 20.1(f) shows that, after the fault injection, Sensor#1 is weighted less than the remaining sensors. This is due to the fact that the residual η_k^1 [see Equation (20.13)] of Sensor#1 in Figure 20.1(c) increases in magnitude with the drift error. The profile of ${}_1w^{\text{rel}}$ in Figure 20.1(f) is governed by its nonlinear relationship with η_k^1 given by Equations (20.15), (20.19) and (20.20). As seen in Figure 20.1(f), ${}_1w^{\text{rel}}$ initially changes very slowly to ensure that it is not sensitive to small fluctuations in sensor data due to spurious noise, such as that resulting from thermal–hydraulic turbulence. The significant reduction in ${}_1w^{\text{rel}}$ takes place after about 32 h and eventually reaches the minimum value of 10^{-3} when η_k^1 is sufficiently large. Therefore, the calibrated estimate $\hat{\boldsymbol{x}}_k$ is practically unaffected by the drifting sensor and stays close to the remaining three healthy sensors. In essence, $\hat{\boldsymbol{x}}_k$ is the average of the three healthy sensors. Upon restoration of Sensor#1 to the normal state, the calibrated signal ${}_1y_k$ temporarily goes down because of the large value of correction ${}_1\hat{c}_k$ at that instant as seen in Figure 20.1(e). However, the adaptive filter quickly brings back ${}_1\hat{c}_k$ to a small value, and thereby the residual ${}_1\eta_k$ is reduced and the original weight (i.e. ~1) is regained. Calibrated and uncalibrated estimates are compared in Figure 20.1(d), which shows a peak difference of about 12°F (6.67°C) over a prolonged period.

In addition to the accuracy of the calibrated estimate, the filter provides fast and smooth recovery from abnormal conditions under both steady-state and transient operations of the power plant. For example, during the transient disturbance after about 65 h, the steam temperature undergoes a relatively large swing. Since the sensors are not spatially collocated, their readings are different during plant transients as a result of transport lag in the steam header. Figure 20.1(f) shows that the weights of two sensors out of the three healthy sensors are temporarily reduced while the remaining healthy sensor

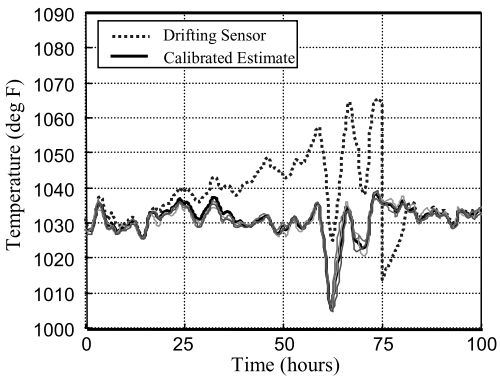
© 2005 by Chapman & Hall/CRC



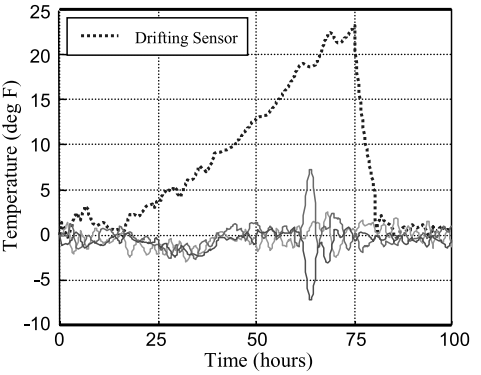
(a) Uncalibrated Sensor Data and the Estimate



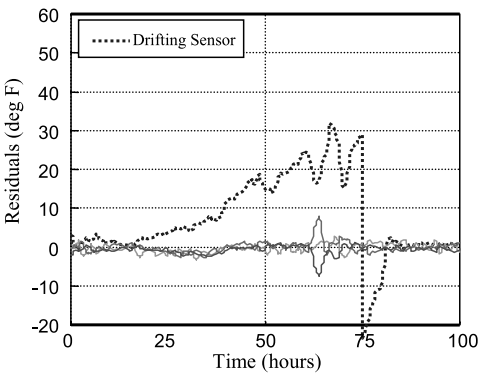
(d) Calibrated and Uncalibrated Estimates



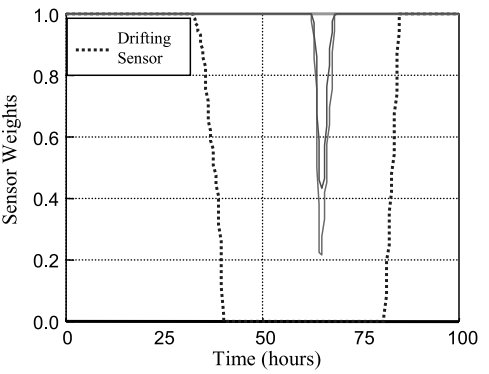
(b) Calibrated Sensor Data and the Estimate



(e) Correction for Sensor Data Calibration



(c) Residuals of the Calibrated Sensor Data



(f) Weights for Sensor Calibration

Figure 20.1. Performance of the calibration filter for drift error in a sensor.

enjoys the full weight and the drifting Sensor#1 has practically no weight. As the transients are over, three healthy sensors resume the full weight. The cause of weight reduction is the relatively large residuals of these two sensors, as seen in Figure 20.1(c). During this period, the two affected sensors undergo modest corrections: one is positive and the other negative, as seen in Figure 20.1(e), so that the calibrated values of the three healthy sensors are clustered together. The health-monitoring system and the plant control system rely on the spatially averaged throttle steam temperature [14–17].

Another important feature of the calibration filter is that it reduces the deviation of the drifting Sensor#1 from the remaining sensors as seen from a comparison of its responses in

© 2005 by Chapman & Hall/CRC

Figure 20.1(a) and (b). This is very important from the perspectives of fault detection and isolation for the following reason. In an uncalibrated system, Sensor#1 might have been isolated as faulty due to accumulation of the drift error. In contrast, the calibrated system makes Sensor#1 temporarily ineffective without eliminating it as faulty. A warning signal can easily be generated when the weight of Sensor#1 diminishes to a small value. This action will draw the attention of maintenance personnel for possible repair or adjustment. Since the estimate $\hat{x}_k$ is not poisoned by the degraded sensor, a larger detection delay can be tolerated. Consequently, the allowable threshold for fault detection can be safely increased to reduce the probability of false alarms.

20.3.3.2 Case 2 (Zero-Mean Fluctuating Error and Recovery in a Single Sensor)

We examine the filter performance by injecting a zero-mean fluctuating error to Sensor#3 starting at 12.5 h and ending at 75 h. The injected error is an additive sine wave of period ~36 h and amplitude 25°F (13.9°C). The simulation results in Figure 20.2 show how the calibration filter responds to the fluctuating error in Sensor#3 while the remaining three sensors (i.e. Sensor#1, Sensor#2 and Sensor#4) are functioning normally. To some extent, the filter response is similar to that of the drift error in Case 1. The major difference is the oscillatory nature of the weights and corrections of Sensor#3, as seen in Figure20.2(f) and (e) respectively. Note that this simulated fault makes the filter switch autonomously to the normal state from either one of the two abnormal states as the sensor error fluctuates between positive and negative limits. Since this is a violation of the Assumption A3 in Appendix A, the recursive relation in Equation (20.A17) represents an approximation of the actual situation. The results in Figure 20.2(b) to (f) show that the filter is sufficiently robust to be able to execute the tasks of sensor calibration and measurement estimation in spite of this approximation. The filter not only exhibits fast response, but also its recovery is rapid regardless of whether the fault is naturally mitigated or corrected by an external agent.

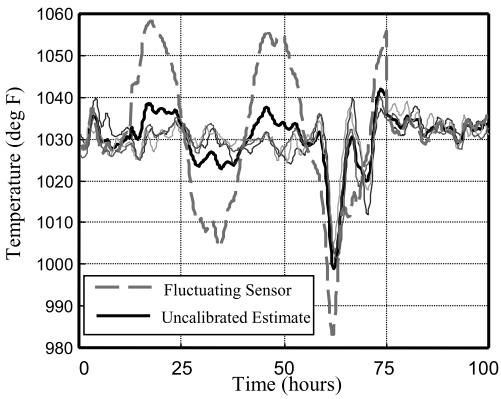
20.3.3.3 Case 3 (Drift Error in One Sensor and Zero-Mean Fluctuating Error in Another Sensor)

This case investigates the filter performance in the presence of simultaneous faults in two out of four sensors. Note that if the two affected sensors have similar types of fault (e.g. common mode faults), the filter will require additional redundancy to augment the information base generated by the remaining healthy sensors. Therefore, we simulate simultaneous dissimilar faults by injecting a drift error in Sensor#1 and a fluctuating error in Sensor#3 exactly identical to those in Case 1 and Case 2 respectively. A comparison of the simulation results in Figure 20.3 with those in Figures 20.1 and 20.2 reveals that the estimate $\hat{x}_k$ is essentially similar in all three cases, except for small differences during the transients at ~65 h. It should be noted that, during the fault injection period from 12.5 to 75 h, $\hat{x}_k$ is strongly dependent on: Sensors#2, #3 and #4 in Case 1; Sensors#1, #2 and #4 in Case 2; and Sensors#2 and #4 in Case 3. Therefore, the estimate $\hat{x}_k$ cannot be exactly identical for these three cases. The important observation in this case study is that the filter can handle simultaneous faults in two out of four sensors provided that these faults are not strongly correlated; otherwise, additional redundancy or equivalent information would be necessary.

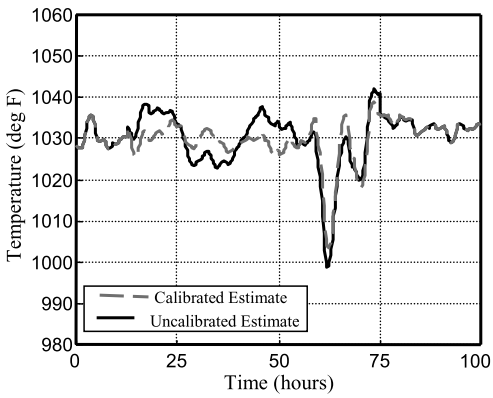
20.4 Summary and Conclusions

This chapter presents a formulation and validation of an adaptive filter for real-time calibration of redundant signals consisting of sensor data and/or analytically derived measurements. Individual signals are calibrated on-line by an additive correction that is generated by a recursive filter. The covariance matrix of the measurement noise is adjusted as a function of the *a posteriori* probabilities of failure of the individual measurements. An estimate of the measured variable is also obtained in real time as a weighted average of the calibrated measurements. These weights are recursively updated in real time instead of being fixed *a priori*. The effects of intra-sample failure and probability of false alarms are

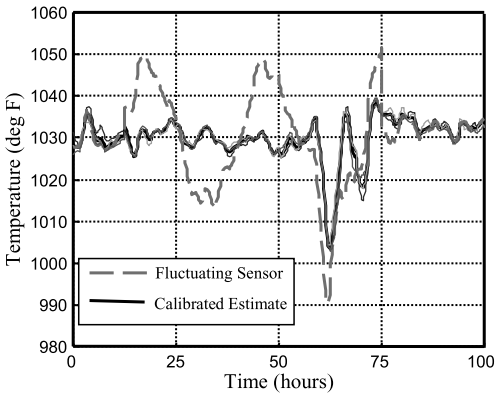
© 2005 by Chapman & Hall/CRC



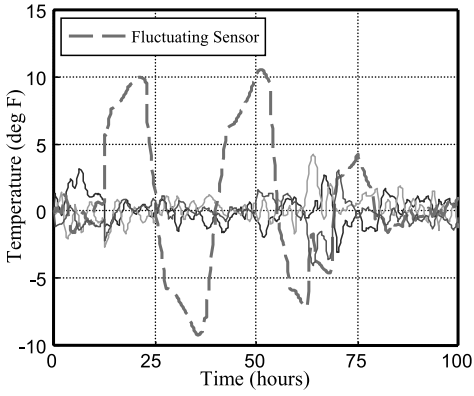
(a) Uncalibrated Sensor Data and the Estimate



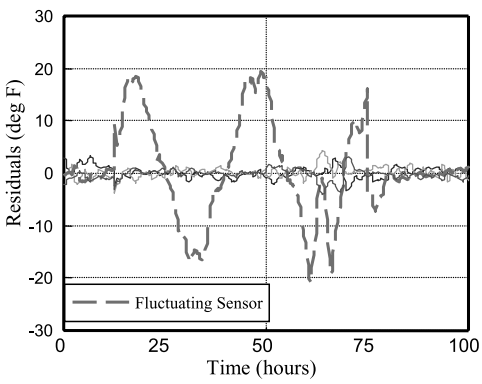
(d) Calibrated and Uncalibrated Estimates



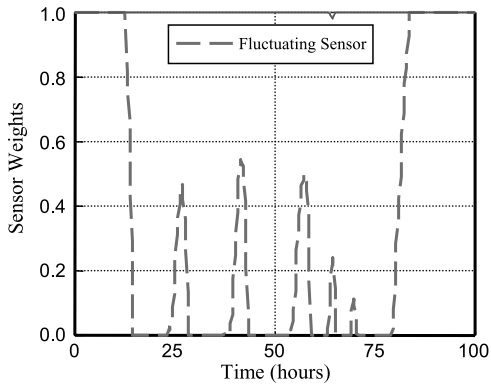
(b) Calibrated Sensor Data and the Estimate



(e) Correction for Sensor Data Calibration



(c) Residuals of the Calibrated Sensor Data



(f) Weights for Sensor Calibration

Figure 20.2. Performance of the calibration filter for fluctuation error in a sensor.

taken into account in the recursive filter. The important features of this real-time adaptive filter are summarized below:

- A model of the physical process is not necessary for calibration and estimation if sufficient redundancy of sensor data and/or analytical measurements is available.
- The calibration algorithm can be executed in conjunction with a fault-detection and isolation system.

© 2005 by Chapman & Hall/CRC

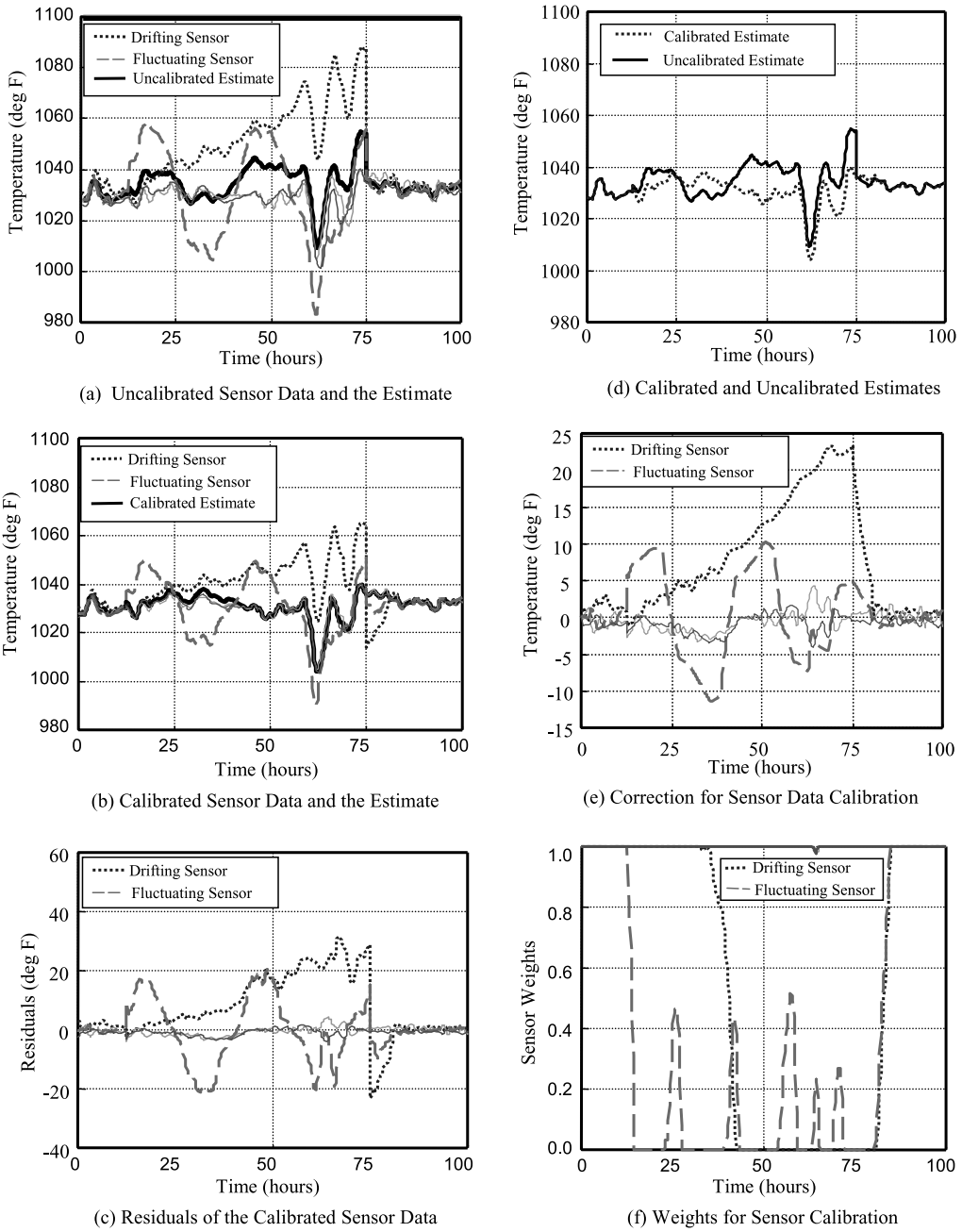


Figure 20.3. Performance of the calibration filter for drift error and fluctuation error in two sensors.

- The filter smoothly calibrates each measurement as a function of its *a posteriori* probability of failure, which is recursively generated based on the current and past observations.

The calibration and estimation filter has been tested by injecting faults in the data set collected from an operating power plant. The filter exhibits speed and accuracy during steady-state and transient operations of the power plant. It also shows fast recovery when the fault is corrected or naturally mitigated. The filter software is portable to any commercial platform and can be potentially used to

© 2005 by Chapman & Hall/CRC

enhance the Instrumentation & Control System Software in tactical and transport aircraft, and nuclear and fossil-fuel power plants.

Appendix A: Multiple Hypotheses Testing Based on Observations of a Single Variable

Let $\{\eta_k, k = 1, 2, 3, \cdots\}$ be (conditionally) independent values of a single variable (e.g. residual of a measurement) at consecutive sampling instants. We assume M distinct possible modes of failure in addition to the normal mode of operation that is designated as the mode 0. Thus, there are $(M+1)$ mutually exclusive and exhaustive hypotheses defined at the kth sample as

$$\begin{aligned} &H_k^0\text{: } \textit{Normal behavior with} \text{ a priori } \textit{density function } f^0(\cdot) \equiv f\left(\cdot \middle| H^0\right) \\ &H_k^i\text{: } \textit{Abnormal behavior with} \text{ a priori } \textit{density function } f^i(\cdot) \equiv f\left(\cdot \middle| H^i\right), i = 1, 2, \ldots, M \end{aligned} \tag{20.A1}$$

where each hypothesis $H_k^j, j = 0, 1, 2, \ldots, M$ is treated as a Markov state.

We define the *a posteriori* probability π_k^j of the jth hypothesis at the kth sample as

$$\pi_k^j \equiv P[H_k^j | Z_k], j = 0, 1, 2, \ldots, M \tag{20.A2}$$

based on the history $Z_k \equiv \cap_{i=1}^k z_i$ where $z_i \equiv \{\eta_i \in B_i\}$ and B_i is the region of interest at the ith sample. The problem is to derive a recursive relation for *a posteriori* probability of failure Π_k at the kth sample:

$$\Pi_k \equiv P\left[\bigcup_{j=1}^M H_k^j | Z_k\right] = \sum_{j=1}^M P\left[H_k^j | Z_k\right] \Rightarrow \Pi_k = \sum_{j=1}^M \pi_k^j \tag{20.A3}$$

because of the exhaustive and mutually exclusive properties of the Markov states, $H_k^j, j = 1, 2, \ldots, M$. To construct a recursive relation for Π_k, we introduce the following three definitions:

$$\text{Joint probability: } \xi_k^j \equiv P[H_k^j, Z_k] \tag{20.A4}$$

$$A \textit{ priori} \text{ probability: } \lambda_k^j \equiv P[z_k | H_k^j.] \tag{20.A5}$$

$$\text{Transition probability: } \alpha_k^{i,j} \equiv P[H_k^j | H_{k-1}^i] \tag{20.A6}$$

Then, because of conditional independence of z_k and Z_{k-1}, Equation (A-4) takes the following form:

$$\begin{aligned} \xi_k^j &= P[H_k^j, z_k, Z_{k-1}] \\ &= P[z_k \left| H_k^j \right] P[H_k^j, Z_{k-1}] \end{aligned} \tag{20.A7}$$

Furthermore, the exhaustive and mutually exclusive properties of the Markov states $H_k^j, j = 0, 1, 2, \ldots, M$, and independence of Z_{k-1} and H_k^j lead to

$$\begin{aligned} P\left[H_k^j, Z_{k-1}\right] &= \sum_{i=0}^M P\left[H_k^j, H_{k-1}^i, Z_{k-1}\right] \\ &= \sum_{i=0}^M P\left[Z_{k-1} \middle| H_{k-1}^i\right] P\left[H_k^j | H_{k-1}^i\right] P\left[H_{k-1}^i\right] \\ &= \sum_{i=0}^M P\left[H_k^j | H_{k-1}^i\right] P\left[H_{k-1}^i, Z_{k-1}\right] \end{aligned} \tag{20.A8}$$

© 2005 by Chapman & Hall/CRC

The following recursive relation is obtained from a combination of Equations (20.A4) to (20.A8) as:

$$\xi_k^j = \lambda_k^j \sum_{i=0}^{M} a_k^{i,j} \xi_{k-1}^i \tag{20.A9}$$

We introduce a new term

$$\psi_k^j \equiv \frac{\xi_k^j}{\xi_k^0} \tag{20.A10}$$

that reduces to the following form by use of Equation (20.A9):

$$\psi_k^j = \left(\frac{\lambda_k^j}{\lambda_k^0}\right)\left(\frac{a_k^{0,j} + \sum_{i=1}^{M} a_k^{i,j}\psi_{k-1}^i}{a_k^{0,0} + \sum_{i=1}^{M} a_k^{i,0}\psi_{k-1}^i}\right) \tag{20.A11}$$

to obtain the *a posteriori* probability π_k^j in Equation (20.A2) in terms of ξ_k^j and ψ_k^j as

$$\begin{aligned}\pi_k^j &= \frac{P[H_k^j, Z_k]}{P[Z_k]} = \frac{P[H_k^j, Z_k]}{\sum_{i=0}^{M} P\left[H_k^i, Z_k\right]} \\ &= \frac{\xi_k^j}{\xi_k^0 + \sum_{i=1}^{M} \xi_k^i} = \frac{\psi_k^j}{1 + \sum_{i=1}^{M} \psi_k^i}\end{aligned} \tag{20.A12}$$

A combination of Equations (20.A3) and (20.A12) leads to the *a posteriori* probability Π_k of failure as

$$\Pi_k = \frac{\Psi_k}{1 + \Psi_k} \quad \text{with} \quad \Psi_k \equiv \sum_{j=1}^{M} \psi_k^j \tag{20.A13}$$

The above expressions can be realized by a simple recurrence relation under the following four assumptions:

- *Assumption A1.* At the starting point (i.e. $k = 0$), all measurements operate in the normal mode, i.e. $P[H_0^0] = 1$ and $P[H_0^j] = 0$ for $j = 1, 2, \ldots, M$. Therefore, $\xi_0^0 = 1$ and $\xi_0^j = 0$ for $j = 1, 2, \ldots, M$.
- *Assumption A2.* Transition from the normal mode to any abnormal mode is equally likely. That is, if p is the *a priori* probability of failure during one sampling interval, then $a_k^{0,0} = 1 - p$ and $a_k^{0,i} = p/M$ for $i = 1, 2, \ldots, M$, and all k.
- *Assumption A3.* No transition takes place from an abnormal mode to the normal mode, implying that $a_k^{i,0} = 0$ for $i = 1, 2, \ldots, M$, and all k. The implication is that a failed sensor does not return to the normal mode (unless replaced or repaired).
- *Assumption A4.* Transition from an abnormal mode to any abnormal mode, including itself, is equally likely. That is, $a_k^{i,j} = 1/M$ for $i, j = 1, 2, \ldots, M$, and all k.

© 2005 by Chapman & Hall/CRC

A recursive relation for Ψ_k is generated based on the above assumptions and using the expression in Equation (20.A11) as

$$\psi_k^j = \frac{p + \sum_{i=1}^{M} \psi_{k-1}^i}{(1-p)M} \left(\frac{\lambda_k^j}{\lambda_k^0}\right) \quad \text{given} \quad \psi_0^j = 0 \quad \text{for} \quad j = 1, 2, \ldots, M \tag{20.A14}$$

which is simplified by use of the relation $\Psi_k \equiv \sum_{i=1}^{M} \psi_k^i$ in Equation (20.A13) as

$$\Psi_k = \left(\frac{p + \Psi_{k-1}}{(1-p)M}\right) \sum_{j=1}^{M} \frac{\lambda_k^j}{\lambda_k^0} \quad \text{given} \quad \Psi_0 = 0 \tag{20.A15}$$

If the probability measure associated with each abnormal mode is absolutely continuous relative to that associated with the normal mode, then the ratio λ_k^j/λ_k^0 of *a priori* probabilities converges to a Radon–Nikodym derivative as the region B_k in the expression $z_k \equiv \{\eta_k \in B_k\}$ approaches zero measure [18]. This Radon–Nikodym derivative is simply the likelihood ratio $f^j(\eta_k)/f^0(\eta_k), j = 1, 2, \ldots, M$, where $f^i(\cdot)$ is the *a priori* density function conditioned on the hypothesis $H^i, i = 0, 1, 2, \ldots, M$.

Accordingly, Equation (20.A15) becomes

$$\Psi_k = \left(\frac{p + \Psi_{k-1}}{(1-p)M}\right) \sum_{j=1}^{M} \frac{f^j(\eta_k)}{f^0(\eta_k)} \quad \text{given} \quad \Psi_0 = 0 \tag{20.A16}$$

For the specific case of two abnormal hypotheses (i.e. $M = 2$) representing positive and negative failures, the recursive relations for Ψ_k and Π_k in Equations (20.A16) and (20.A13) become

$$\left.\begin{aligned} \Psi_k &= \left(\frac{p + \Psi_{k-1}}{2(1-p)}\right)\left(\frac{f^1(\eta_k) + f^2(\eta_k)}{f^0(\eta_k)}\right) \\ \Pi_k &= \frac{\Psi_k}{1 + \Psi_k} \end{aligned}\right\} \quad \text{given} \quad \Psi_0 = 0 \tag{20.A17}$$

References

[1] Dickson, B. *et al.*, Usage and structural life monitoring with HUMS, in *American Helicopter Society 52nd Annual Forum*, Washington, DC, June 4–6, 1996, 1377.

[2] Potter, J.E. and Suman, M.C., Thresholdless redundancy management with arrays of skewed instruments, in Integrity in Electronic Flight Control Systems, NATO AGARDOGRAPH-224, 1977, 15-1.

[3] Daly, K.C. *et al.*, Generalized likelihood test for FDI in redundant sensor configurations, *Journal of Guidance and Control*, 2(1), 9, 1979.

[4] Ray, A. *et al.*, Analytic redundancy for on-line fault diagnosis in a nuclear reactor, *AIAA Journal of Energy*, 7(4), 367, 1983.

[5] Deckert, J.C. *et al.*, Signal validation for nuclear power plants, *ASME Journal of Dynamic Systems, Measurement and Control*, 105(1), 24, 1983.

[6] Desai, M.N. *et al.*, Dual sensor identification using analytic redundancy, *Journal of Guidance and Control*, 2(3), 213, 1979.

[7] Ray, A. and Desai, M., A redundancy management procedure for fault detection and isolation, *ASME Journal of Dynamic Systems, Measurement and Control*, 108(3), 248, 1986.

© 2005 by Chapman & Hall/CRC

[8] Ray, A. and Phoha, S., Detection and identification of potential faults via multilevel hypothesis testing, *Signal Processing*, 82, 853, 2002.
[9] Ray, A. and Luck, R., Signal validation in multiply-redundant systems, *IEEE Control Systems Magazine*, 11(2), 44, 1996.
[10] Jazwinski, A.H., *Stochastic Processes and Filtering Theory*, Academic Press, New York, 1970.
[11] Gelb, A. (ed.), *Applied Optimal Estimation*, MIT Press, Cambridge, MA, 1974.
[12] Ray, A., Sequential testing for fault detection in multiply-redundant systems, *ASME Journal of Dynamic Systems, Measurement and Control*, 111(2), 329, 1989.
[13] Stultz, S.C. and Kitto, J.B. (eds), *Steam: Its Generation and Use*, 40th ed., Babcock & Wilcox Co., Baberton, OH, 1992.
[14] Kallappa, P.T. *et al.*, Life extending control of fossil power plants for structural durability and high performance, *Automatica*, 33(6), 1101, 1997.
[15] Kallappa, P.T. and Ray, A., Fuzzy wide-range control of fossil power plants for life extension and robust performance, *Automatica*, 36(1), 69, 2000.
[16] Holmes, M. and Ray, A., Fuzzy damage mitigating control of mechanical structures, *ASME Journal of Dynamic Systems, Measurement and Control*, 120(2), 249, 1998.
[17] Holmes, M. and Ray, A., Fuzzy damage mitigating control of a fossil power plant, *IEEE Transactions on Control Systems Technology*, 9(1), 140, 2001.
[18] Wong, E. and Hajek, B., *Stochastic Processes in Engineering Systems*, Springer-Verlag, New York, 1985.

© 2005 by Chapman & Hall/CRC

21

Semantic Information Extraction

David S. Friedlander

21.1 Introduction

This chapter describes techniques for extracting *semantic* information from sensor networks and applying them to recognizing the behaviors of autonomous vehicles based on their trajectories, and predicting anomalies in mechanical systems based on a network of embedded sensors.

Sensor networks generally observe systems that are too complex to be simulated by computer models based directly on their physics. We therefore use a semi-empirical model based on time series measurements. These systems can be stochastic, but not necessarily stationary. In this case, we make a simplifying assumption, that the time scale for changes in the equations of motion is much longer than the time scale for changes in the dynamical variables. The system is then in semi-equilibrium, so its dynamics can be determined from a data sample whose duration is long compared with changes in the dynamical variables but short compared with changes in the dynamical equations.

The techniques are based on integrating and converting sensor measurements into formal languages and using a *formal language measure* to compare the language of the observations with the languages associated with known behaviors stored in a database. Based on the hypothesis that behaviors represented by similar formal languages are semantically similar, this method provides a form of computer perception for physical behaviors through the extension of traditional pattern-matching techniques.

One intriguing aspect of this approach is that people represent their perception of the environment with *natural language*. Statistical approaches to analyzing formal languages have been successfully applied to *natural language processing* (NLP) [1]. This suggests that formal languages may be a promising approach for representing sensor network data.

21.2 Symbolic Dynamics

In symbolic dynamics, the numeric time series associated with a system's dynamics are converted into streams of symbols. The streams define a formal language where any substring in the stream belongs to the language. Conversions of physical measurements to symbolic dynamics and the analysis of the resulting strings of symbols have been used for characterizing nonlinear dynamical systems as they

© 2005 by Chapman & Hall/CRC

simplify data handling while retaining important qualitative phenomena. This also allows usage of complexity measures defined on formal languages made of symbol strings to characterize the system dynamics [2]. The distance between individual symbols is not defined, so there is no notion of linearity.

21.2.1 The Conversion of System Dynamics into Formal Languages

One method for generating a stream of symbols from the resampled sensor network data divides the phase-space volume of the network into hypercube-shaped regions and assigns a symbol to each region. When the phase-space trajectory enters a region, its symbol is added to the symbol stream as shown in Figure 21.1.

Any set containing strings of symbols defines a formal language. If the language contains an infinite number of strings, then it cannot be fully represented in this way. The specification of a formal language can be compressed, allowing finite representations of infinite languages. Usually, greater compression provides greater insight into the language. Two equivalent representations are generally used: finite-state machines and formal grammars.

Chomsky [3] developed a classification of formal languages based on their complexity. From least to most complex, they are: regular, context free, context sensitive and recursively enumerable. The simplest is regular languages, which can be represented by finite-state automata. Since the dynamics of complex systems are generally stochastic, we use probabilistic finite-state automata (PFSA).

Sometimes PFSA for very simple systems can be specified intuitively. There is also a method to determine them analytically [4]. Finite-state machines determined this way are called ε-machines. Unfortunately, the method is currently limited to regular languages.

21.2.2 Determination of ε-Machines

The symbol stream is converted into a PFSA or, equivalently, a probabilistic regular language. A sample of the symbol stream of some length L is used to determine the model. Shalizi's method generates a PFSA from the sample. Since the symbol stream is unbounded, each state is considered an accepting state. Each state s in the PFSA is assigned a set of substrings U, such that the path of any string that is accepted by the automaton and ends in $u \in U$ will be accepted at state s.

Each state also contains a *morph*, which is a list of the probabilities of each symbol being emitted from that state, i.e. $\bar{M}(S_i)$: $M_j(S_i) \equiv P(e_j|S_i)$, where $\bar{M}(S_i)$ is the morph of state S_i and $P(e_j|S_i)$ is the probability of emitting the jth symbol e_j when the system is in state S_i. The probabilities are

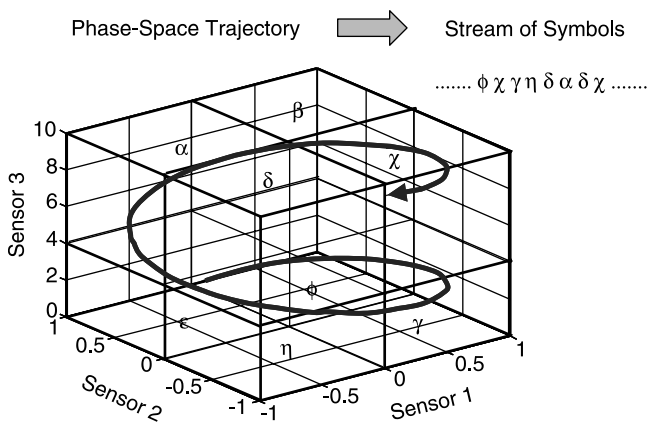


Figure 21.1. Continuous to symbolic dynamics.

© 2005 by Chapman & Hall/CRC

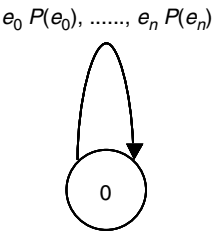


Figure 21.2. Initial automaton.

approximated by the statistics of the sample. Let $U^i \equiv \{s_k^i\}$ be the set of substrings assigned to state S_i. The quantity

$$M_j(S_i) \approx \sum_k \frac{\left|e_j s_k^i\right|}{\left|s_k^i\right|}$$

where $|s_l|$ is the count of the substring s_l in the sample and new symbols are appended on the left-hand side of strings.

The PFSA is initialized to a single state containing the empty string. Its morph is $\bar{M}(S_0) = \{P(e_i)\}$, where S_0 is the initial state and the ith component of $\bar{M}(S_0)$ is the unconditional probability of symbol e_i. Then add transitions $S_0 \xrightarrow{e_i, P(e_i)} S_0$ for each symbol e_i. In other words, the initial morph contains the probability of each individual symbol. The initial automaton is shown in Figure 21.2.

The initial automaton is expanded using the Shalizi *et al.* algorithim [4]. A simplified version is given in Figure 21.3. Strings of length one through some maximum length are added to the PFSA. Given a state, containing string s, the string $s' = e_k||s$, where e_k is the kth symbol and "||" is the concatenation operator, will have the morph $\bar{M}(s') = \{P(e_k|s')\}$. If an existing state $\hat{S}$ has a morph close to $\bar{M}(S)$, the transition $S \xrightarrow{e_i} \hat{S}$ is added to the PFSA and the string s' is added to $\hat{S}$. Otherwise, a new state $\tilde{S}$ and the transition $S \xrightarrow{e_i} \tilde{S}$ are added to the PFSA. The string s' and the morph $\bar{M}(s')$ are assigned to $\tilde{S}$.

The next stage is to *determinize* [4] the PFSA by systematically adding states whenever a given state has two or more transitions leaving it with the same symbol. Finally, the *transient states* are eliminated, i.e. a state is transient if it cannot be reached from any other state.

Most complexity measures are based on entropy and, therefore, are a minimum for constant data streams and a maximum for random data streams. This, however, contradicts the intuitive notion of complexity, which is low for both constant and random behavior of dynamical systems (see Figure 21.2). Crutchfield [5] introduced a measure called ε-complexity that is defined based on the construction of a PFSA for the symbol stream. The ε-complexity is defined as the Shannon entropy of the state probabilities of the automaton: $C_\varepsilon \equiv \sum_i P(S_i) \log P(S_i)$. It is minimal for both constant and random behavior and diverges when chaotic behavior is exhibited, i.e. the number of states in the PSFA goes to infinity as some system parameter goes to its critical value for chaotic behavior.

21.3 Formal Language Measures

One shortcoming of complexity measures for detecting, predicting or classifying anomalous behaviors is that they are scalars. That is, two different behaviors of a complex system may have the same complexity measure.

Ray and Phoha [6] have addressed this problem by representing each possible formal language with a given alphabet Σ. The language of all possible strings is denoted as $\Sigma^\star$. It is represented as the unit vector in an infinite-dimensional vector space $\left(2^{\Sigma^\star}, \oplus\right)$ over the finite field $GF(2)$ where $\oplus$ is the exclusive-OR operator for vector addition and the zero vector in this space is the null language $\emptyset$.

There are at least two methods to determine $\mu(L)$, the measure of language L. If a PFSA can be derived for the language, then an exact measure developed Wang and Ray [7] can be used. If the

© 2005 by Chapman & Hall/CRC

```
Function Automaton (string-sample, S0, {transitions})
    for each L = 2, Lmax
        for each state, Si
            for each string s_i^k ∈ Si
                if | s_i^k | + 1 = L
                    for each symbol ej
                        M̄ (s_i^k ej) = morph (s_i^k ej)
                        if M̄ (s_i^k ej) ≈ M̄ (si)
                            add-string-to-state (s_i^k ej, Si,Si, {transitions})
                        elseif M̄ (s_i^k ej) ≈ M̄ (Sm), where m ≠ i
                            add-string-to-state (s_i^k ej, Si,Sm, {transitions})
                        else
                            create new state Sn
                            add-string-to-state (s_i^k ej, Si,Sn, {transitions})
                        end
                    end
                end
            end
        end
    end
    return {Si}, {transitions}

Function morph (string)
    for each symbol ej
        Mj(string) = |string| / |string ||ej|
    end
    return M̄ (string)

Function add-string-to-state (string, S0, Sf, {transitions})
    add string to Sf
    ek = last-symbol (string)
    add S0 --ek--> Sf to {transitions}
```

Figure 21.3. Algorithm for building ε-machine.

language is not regular and cannot be well approximated by a PFSA, then an approximate measure developed by the author can be used instead. In either case, the distance between two formal languages, L_1 and L_2, is defined as $d(L_1, L_2) \equiv \left|\mu(L_1 \cup L_2 - L_1 \cap L_2)\right|$, i.e. the measure of the exclusive-OR of the strings in the languages. The only restriction on the two languages is that they come from the same alphabet of symbols. In other words, the two languages must represent dynamical processes defined on the same phase space.

The measures can be applied to a single language or to the vector difference between any two languages where the vector difference corresponds to the exclusive-OR operation of the strings belonging to the languages. Since the exclusive-OR of the language vectors maps back to the *symmetric set difference* of the languages, this vector addition operation can be considered as taking the *difference* between two languages.

Friedlander *et al.* [8] have proposed another measure that is a real positive measure μ: $2^{\Sigma^*} \rightarrow [0, \infty)$, called the *weighted counting measure*, defined as

$$\mu(L) \equiv \sum_{i=1}^{\infty} w_i n_i(L)$$

© 2005 by Chapman & Hall/CRC

where $n_i(L)$ is the number of strings of length ℓ in the language L, and $w_i = \frac{1}{(2k)^i}$ where the positive integer $k = |\Sigma|$ is the alphabet length. The weighting factor w_i was designed so that $\mu(\Sigma^*) = 1$. The weighting factor w_i decays exponentially with the string length ℓ. This feature allows good approximations to the language measure from a relatively small sample of a language with a large number of strings.

21.4 Behavior Recognition

If we define a *behavior* as a pattern of activity in the system dynamics, and represent it as a formal language, we can compare an observed behavior with a database of known behaviors and determine the closest match using a distance based on a formal language measure [9]. We can also discover new behaviors that are based on clusters of formal language vectors. When the behavior is based on an object's trajectory, the techniques can be applied to surveillance and defense.

The concepts of ε-machines, language measures, and distance functions can be used to apply traditional pattern-matching techniques to behavior recognition. For example, we can store a set of languages $\{L_i\}$ corresponding to known behaviors and use them as exemplars. When the sensor network records some unknown target behavior with language L_u, it can be compared with the database to find the best matching language of some known behavior using

$$\text{Behavior}(L_k)\text{: } k = \text{index} \max_i d(L_u, L_i).$$

Target behaviors will change over time, and it is desirable to track those changes as they occur. This can be done with the method presented here as long as the time scale for detecting behaviors, i.e. the length of the language sample, is shorter that the time scale for behavior changes. The sensor data are sampled at regular intervals, the behavior for each interval is determined, and changes in the corresponding languages can be analyzed.

If we define an *anomaly* as an abrupt and significant change in system dynamics, it will include faults (recoverable errsors) and failures (unrecoverable errors). When the behaviors are based on anomaly precursors, the technique can be applied to *condition-based maintenance,* providing early prediction of failures in mechanical systems. Taking corrective action in advance could increase safety, reliability, and performance.

21.5 Experimental Verification

This section contains the results of early experiments that test our method for extracting semantic information from sensor network data. We attempted to distinguish between two types of robot behavior: following a perimeter and a random search. This system was consistently able to recognize the correct behavior and detect changes from one behavior to the other. Although preliminary, these results suggest the new methods are promising.

The experiments use a pressure-sensitive floor measuring simple, single robot behaviors [9]. Owing to the noisiness and unreliability of the pressure sensors, they were used only to determine the quadrant of the floor where the robot was located. The results show the robustness of our technique. Pressure-sensitive wire was placed under ''panels'' of either 2×2 or 2×1 square floor tiles that were 584 mm on a side.

Each panel is numbered in the diagram. The floor is divided into four quadrants (as shown in Figure 21.4). The panels 12, 13, 14, 15, 16, 25, and 3 are between quadrants and their data were not used in the experiment. Each panel is a sensor and provides time series data that were analyzed in real time. The upper-left quadrant had seven panels and the others had five each. This redundancy provided experimental robustness while using unreliable and noisy sensing devices. One or more of the panels did not work, or worked incorrectly during most of the experimental runs.

© 2005 by Chapman & Hall/CRC

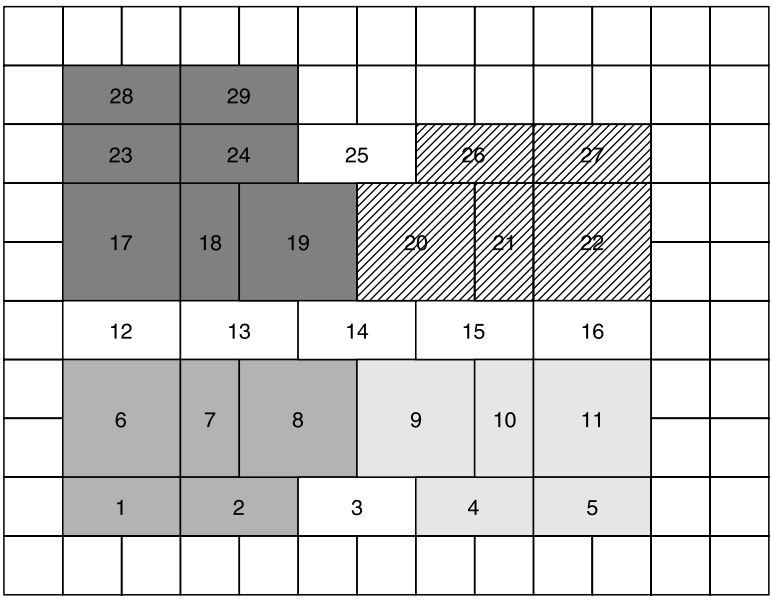


Figure 21.4. Pressure sensitive floor.

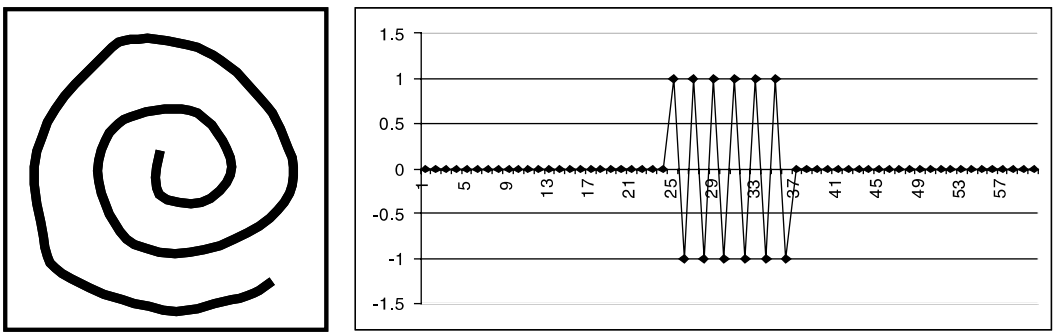


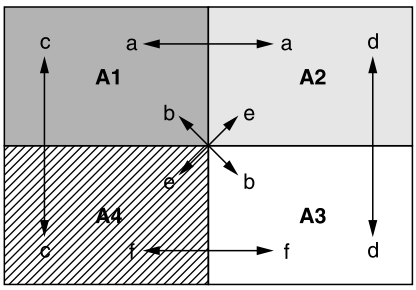
Figure 21.5. Pressure sensor data: (a) pressure panel: (b) data sample.

The experiments involved dynamically differentiating between *wall-following* and *random search* behaviors of a single robot. The sensors were built by coiling pressure-sensitive wire under each panel, as shown in Figure 21.5. The robot had four wheels that passed over multiple points in the coiled wire as it ran over the panel. Each panel provided a series of data peaks as the robot crossed over it.

The first step in processing the 29 channels of time series data was to localize the robot in terms of which panel it was crossing when the data sample was taken. Two unsynchronized servers, one for panels 1 to 16 and the other for panels 17 to 29 provided the data. The data were *pushed,* in the sense that a server provided a sample whenever one or more of the panels had an absolute value over an adjustable cutoff.

When the real-time behavior-recognition software receives a data packet, it preprocesses the data. The first stage is to remove the data from the panels between quadrants. If there is no large value for any of the remaining panels, then the packet is ignored; otherwise, the panel with the highest absolute value is considered the location of the robot. This transforms the time series data into a symbol stream of panel id numbers. The next stage filters the panel number stream to reduce noise. The stream values

© 2005 by Chapman & Hall/CRC



Symbol	Event description
a	A1 to A2 or A2 to A1
b	A1 to A3 or A3 to A1
c	A1 to A4 or A4 to A1
d	A2 to A3 or A3 to A2
e	A4 to A2 or A2 to A4
f	A4 to A3 or A3 to A4

Figure 21.6. Event definitions for circling behavior.

flow into a buffer of length 5. Whenever the buffer is full of identical panel numbers this number is emitted to the filtered stream; if an inconsistent number enters the buffer, the buffer is flushed. This eliminates false positives by requiring five peaks per panel.

The panel id stream is then converted to a stream of quadrants, as shown in Figures 21.4 and 21.6. The stream of quadrants is then converted into a stream of events. An event occurs when the robot changes quadrants, as shown in Figure 21.6. The event depends only on the two quadrants involved, not the order in which they are crossed. This was done to lower the number of symbols in the alphabet from 12 to 6.

The next stage is to use the event stream to recognize behaviors. Because the language of wall following is a subset of the language of random searching, the event stream can prove the random search and disprove the wall-following hypotheses, but not prove the wall following and disprove the random-search hypotheses. The longer the event stream is recognized by the wall-following automaton, however, the more evidence there is that the robot is wall following rather than performing a random search. The finite-state automaton in Figure 21.7 recognizes the stream of symbols from wall-following behavior, starting in any quadrant and going in either direction.

The initial behavior is given as *unknown* and goes to *wall following* or *random walk*. It then goes between these two behaviors during the course of the experiment, depending on the frequency of string rejections in the wall-following automaton. If there is less than one string rejection for every six events, then the behavior is estimated to be *wall following*, otherwise it is estimated to be *random walk*.

The displays associated with the behavior-recognition software demonstration are shown in Figure 21.8. There are four displays. The behavior-recognition software shows the current estimate of

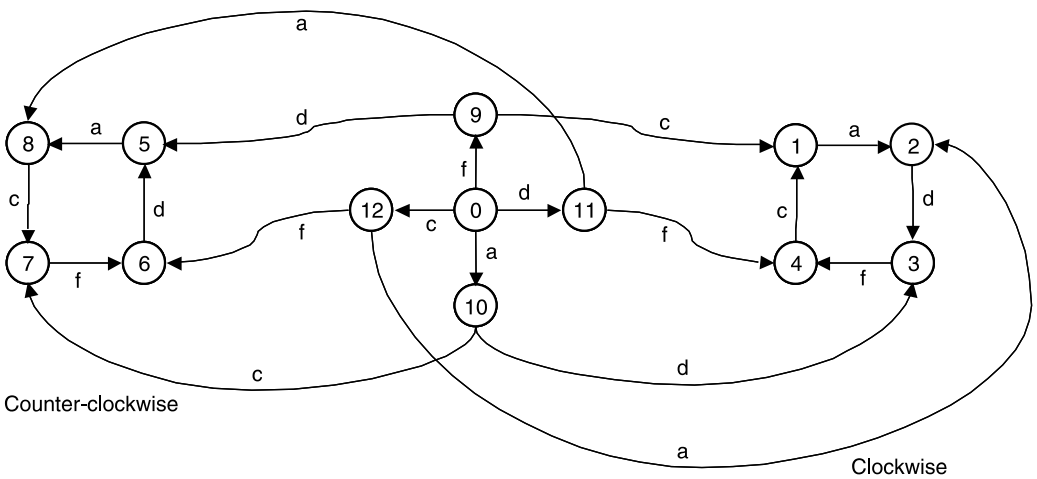


Figure 21.7. Automaton to recognize circling behavior.

© 2005 by Chapman & Hall/CRC

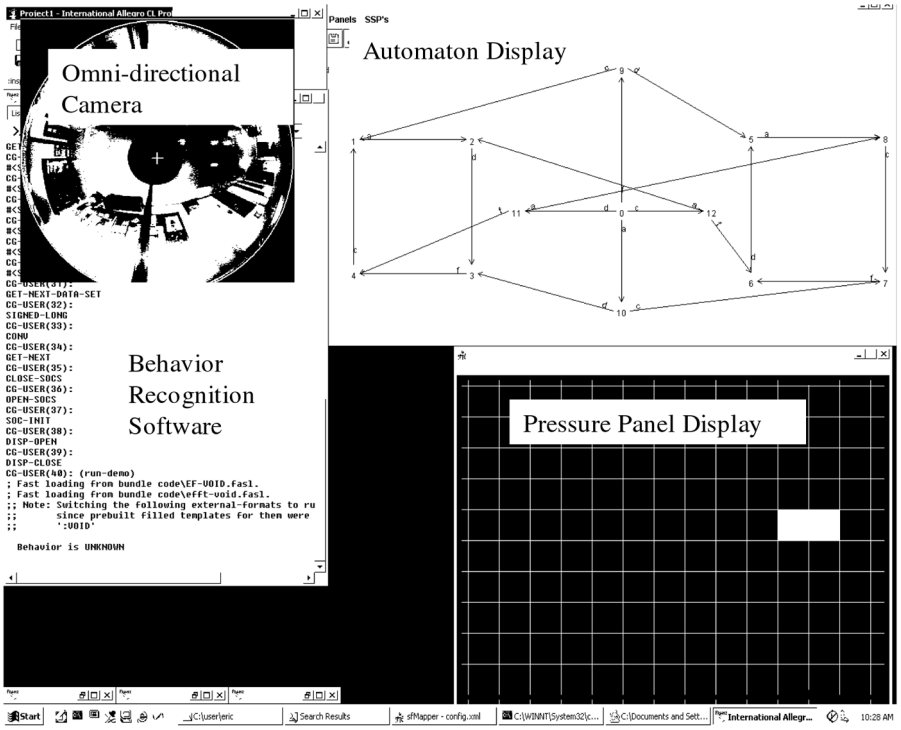


Figure 21.8. Behavior recognition demonstration displays.

the robot's behavior, the symbol and time of each event, and the time of each string rejection. The omni-directional camera shows the physical location of the robot, and the floor panel display shows the panel being excited by the robot. The automaton display shows the current state of the wall-following model based on the event stream.

21.6 Conclusions and Future Work

Traditional pattern-matching techniques measure the distances between the feature vectors of an observed object and a set of stored exemplars. Our research extends these techniques to dynamical systems using symbolic dynamics and a recent advance in formal language theory defining a formal language measure. It is based on a combination of nonlinear systems theory and language theory. It is assumed that mechanical systems under consideration exhibit nonlinear dynamical behavior on two time scales. Anomalies occur on a *slow time scale* that is at least two or more orders of magnitude larger than the *fast time scale* of the system dynamics. It is also assumed that the dynamical system is stationary at the fast time scale and that any nonstationarity is observable only on the slow time scale.

Finite-state machine representations of complex, nonlinear systems have had success in capturing essential features of a process while leaving out irrelevant details [8,10]. These results suggest that the behavior-recognition mechanism would be effective in artificial perception of scenes and actions from sensor data. Applications could include image understanding, voice recognition, fault prediction, and intelligent control. We have experimentally verified the method for two simple behaviors.

Future research may include collection and analysis of additional data on gearboxes and other mechanical systems of practical significance. Another area of future research is integration of formal language measures into damage-mitigating control systems. The methods should be tested on behaviors that are more complex. We are in the process of analyzing data from observations of such behaviors

© 2005 by Chapman & Hall/CRC

using LADAR. They include coordinated actions of multiple robots. One planned experiment contains behavior specifically designed to create a context free, but not regular, language.

Another application is data compression. The definition of the formal language describing the observation is transmitted, rather than the sensor data itself. At the receiving end, the language can be used to classify the behavior or to regenerate sensor data that are statistically equivalent to the original observations. We have begun research to develop this technique in the context of video data from a wireless network of distributed cameras.

Acknowledgments and Disclaimer

This material is based upon work supported in part by the ESP MURI Grant No. DAAD19-01-1-0504; and by the NASA Glenn Research Center under Grant No. NAG3-2448. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the Defense Advanced Research Projects Agency (DARPA), the Army Research Office, or the NASA Glenn Research Center.

References

[1] Charniak, E., *Statistical Language Learning*, MIT Press, Cambridge, MA, 1993.

[2] Kurths, J. *et al.*, Measures of complexity in signal analysis, in *3rd Technical Conference on Nonlinear Dynamics (Chaos) and Full Spectrum Processing*, July 10–13, Mystic, CT, New London, CT, 1995.

[3] Chomsky, N., *Syntactic Structures*, Mouton, Gravenhag, 1957.

[4] Shalizi, C.R. *et al.*, An algorithm for pattern discovery in time series, Santa Fe Institute Working Paper 02-10-060, 2002. Available at http://arxiv.org/abs/cs.LG/0210025.

[5] Crutchfield, J.P., The calculi of emergence, dynamics, and induction, *Physica D,* 75, 11, 1994.

[6] Ray, A. and Phoha, S., A language measure for discrete-event automata, in *Proeedings of the International Federation of Automatic Control (IFAC) World Congress b'02*, Barcelona, Spain, July, 2002.

[7] Wang, X. and Ray, A., Signed real measure of regular languages, in *Proceedings of the American Control Conference*, Anchorage, AK, May, 2002.

[8] Friedlander, D.S. *et al.*, Anomaly prediction in mechanical system using symbolic dynamics, in *Proceedings of the American Control Conference,* Boulder, CO, 2003.

[9] Friedlander, D.S. *et al.*, Determination of vehicle behavior based on distributed sensor network data, in *Proceedings of SPIE's 48th Annual Meeting*, San Deigo, CA, 3–8 August, 2003 (to be published).

[10] Shalizi, C.R., Causal architecture, complexity, and self-organization in time series and cellular automata, Ph.D. dissertation, Physics Department, University of Wisconsin-Madison, 2001.

© 2005 by Chapman & Hall/CRC

22

Fusion in the Context of Information Theory

Mohiuddin Ahmed and Gregory Pottie

22.1 Introduction

In this chapter we selectively explore some aspects of the theoretical framework that has been developed to analyze the nature, performance, and fundamental limits for information processing in the context of data fusion. In particular, we discuss how Bayesian methods for distributed data fusion can be interpreted from the point of view information theory. Consequently, information theory can provide a common framework for distributed detection and communication tasks in sensor networks.

Initially, the context is established for considering distributed networks as efficient information processing entities (Section 22.2). Next, in Section 22.3, the approaches taken towards analyzing such systems and the path leading towards the modern information theoretic framework for information processing are discussed. The details of the mathematical method are highlighted in Section 22.4, and applied specifically for the case of multi-sensor systems in Section 22.5. Finally, our conclusions are presented in Section 22.6.

22.2 Information Processing in Distributed Networks

Distributed networks of sensors and communication devices provide the ability to electronically network together what were previously isolated islands of information sources and sinks, or, more generally, *states of nature*. The states can be measurements of physical parameters (e.g. temperature, humidity, etc.) or estimates of operational conditions (network loads, throughput, etc.), among other things, distributed over a region in time and/or space. Previously, the aggregation, fusion and interpretation of this mass of data representing some phenomena of interest were performed by isolated *sensors*, requiring human supervision and control. However, with the advent of powerful hardware platforms and networking technologies, the possibility and advantages of *distributed sensing* information processing has been recognized [1].

© 2005 by Chapman & Hall/CRC

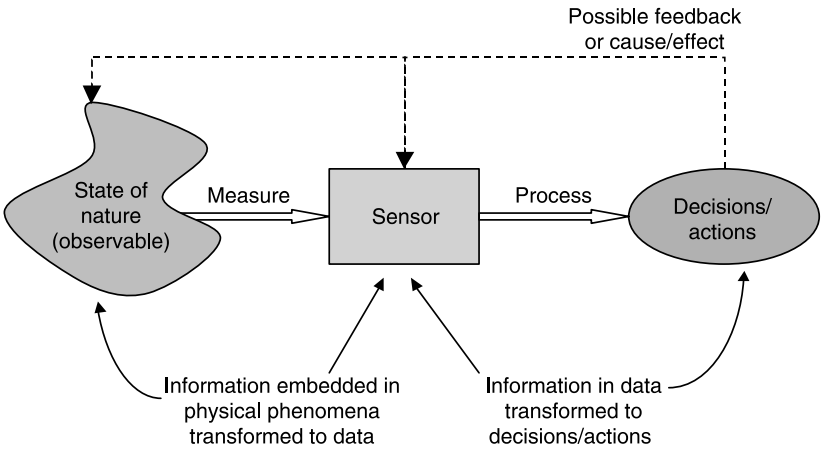


Figure 22.1. Information processing in sensors.

A sensor can be defined to be any device that provides a quantifiable set of outputs in response to a specific set of inputs. These outputs are useful if they can be mapped to a state of nature that is under consideration. The end goal of the sensing task is to acquire a description of the external world, predicated upon which can be a series of *actions*. In this context, sensors can be thought of as *information gathering*, *processing*, *and dissemination* entities, as shown in Figure 22.1. The data pathways in the figure illustrate an abstraction of the flow of information in the system. In a distributed network of sensors, the sensing system may be comprised of multiple sensors that are physically disjoint or distributed in time or space, and that work cooperatively. Compared with a single sensor platform, a network has the advantages of *diversity* (different sensors offer complementary viewpoints), and *redundancy* (reliability and increased resolution of the measured quantity) [2]. In fact, it has been rigorously established from the theory of distributed detection that higher reliability and lower probability of detection error can be achieved when observation data from multiple, distributed sources are intelligently fused in a decision-making algorithm, rather than using a single observation data set [3]. Intuitively, any practical sensing device has limitations on its sensing capabilities (e.g. resolution, bandwidth, efficiency, etc.). Thus, descriptions built on the data sensed by a single device are only approximations of the true state of nature. Such approximations are often made worse by incomplete knowledge and understanding of the environment that is being sensed and its interaction with the sensor. These uncertainties, coupled with the practical reality of occasional sensor failure, greatly compromises reliability and reduce confidence in sensor measurements. Also, the spatial and physical limitations of sensor devices often mean that only partial information can be provided by a single sensor.

A network of sensors overcomes many of the shortcomings of a single sensor. However, new problems in efficient information management arise. These may be categorized into two broad areas [4]:

1. *Data fusion*. This is the problem of combining diverse and sometimes conflicting information provided by sensors in a multi-sensor system in a consistent and coherent manner. The objective is to infer the relevant states of the system that is being observed or activity being performed.
2. *Resource administration*. This relates to the task of optimally configuring, coordinating, and utilizing the available sensor resources, often in a dynamic, adaptive environment. The objective is to ensure efficient[1] use of the sensor platform for the task at hand.

[1]*Efficiency*, in this context, is very general and can refer to power, bandwidth, overhead, throughput, or a variety of other performance metrics, depending upon the particular application.

© 2005 by Chapman & Hall/CRC

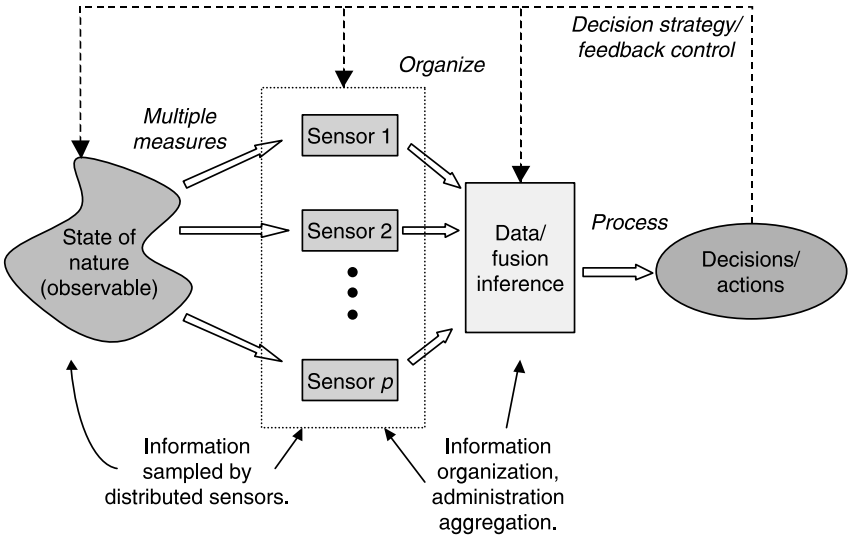


Figure 22.2. Information processing in distributed sensors.

As with the lumped-parameter sensor systems shown in Figure 22.1, the issues mentioned above for multi-sensor systems are shown in Figure 22.2 [2].

22.3 Evolution Towards Information-Theoretic Methods for Data Fusion

Most of the early research effort in probabilistic and information-theoretic methods for data fusion focused on techniques motivated by specific applications, such as in vision systems, sonar, robotics platforms, etc. [5–8]. As the inherent advantages of using multi-sensor systems were recognized [9,10], a need for a comprehensive theory of the associated problems of distributed, decentralized data fusion, and multi-user information theory became apparent [11–13]. Advances in integrated circuit technology have enabled mass production of sensors, signal-processing elements, and radios [14,15], spurring new research in wireless communications [16], and in *ad hoc networking* [17,18]. Subsequently, it was only natural to combine these two disciplines — sensors and networking — to develop a new generation of distributed sensing devices that can work cooperatively to exploit diversity [1,19]. An abridged overview of the research in sensor fusion and management is now given [2].

22.3.1 Sensor Fusion Research

Data fusion is the process by which data from a multitude of sensors are used to yield an optimal estimate of a specified state vector pertaining to the observed system [3], whereas sensor administration is the design of communication and control mechanisms for the efficient use of distributed sensors, with regard to power, performance, reliability, etc. Data fusion and sensor administration have mostly been addressed separately. Sensor administration has been addressed in the context of wireless networking, and not necessarily in conjunction with the unique constraints imposed by data fusion methodologies.

To begin with, sensor models have been aimed at interpretation of measurements. This approach to modeling can be seen in the sensor models used by Kuc and Siegel [5], among others. Probability theory, and in particular a Bayesian treatment of data fusion [20], is arguably the most widely used method for describing uncertainty in a way that abstracts from a sensor's physical and operational details. Qualitative methods have also been used to describe sensors, e.g. by Flynn [21] for sonar and

© 2005 by Chapman & Hall/CRC

infrared applications. Much work has also been done in developing methods for intelligently combining information from different sensors. The basic approach has been to pool the information using what are essentially "weighted averaging" techniques of varying degrees of complexity. For example, Berger *et al.* [10] discuss a majority voting technique based on a probabilistic representation of information. Nonprobabilistic methods [22] used inferential techniques, e.g. for multi-sensor target identification. Inferring the state of nature given a probabilistic representation is, in general, a well understood problem in classical estimation. Representative methods are Bayesian estimation, least squares estimation, Kalman filtering, and its various derivatives. However, the question of how to use these techniques in a distributed fashion has not been addressed to date in a systematic fashion, except for some specific physical-layer cases [23].

22.3.2 Sensor Administration Research

In the area of sensor network administration, protocol development and management have mostly been addressed using application-specific descriptive techniques for specialized systems [9]. Radar tracking systems provided the impetus for much of the early work. Later, robotic applications led to the development of models for sensor behavior and performance that could then be used to analyze and manage the transfer of sensor data. The centralized or hierarchical nature of such systems enabled this approach to succeed. Other schemes that found widespread use were based on determining cost functions and performance trade-offs *a priori* [24], e.g. cost–benefit assignment matrices allocating sensors to targets, or Boolean matrices characterizing sensor–target assignments based on sensor availability and capacity. Expert-system approaches have also been used, as well as decision-theoretic (*normative*) techniques. However, optimal sensor administration in this way has been shown by Tsitsiklis [11] to be very hard in the general framework of distributed sensors, and practical schemes use a mixture of heuristic techniques (e.g. in data fusion systems involving wired sensors in combat aircraft). Only recently have the general networking issues for wireless *ad hoc* networks been addressed [25,26], where the main problems of self-organization, bootstrap, route discovery, etc. have been identified. Application-specific studies, e.g. in the context of antenna arrays [27], have also discussed these issues. However, few general fusion rules or data aggregation models for networked sensors have been proposed, with little analytical or quantitative emphasis. Most of these studies do not analyze in detail the issues regarding the network-global impact of administration decisions, such as choice of fusion nodes, path/tree selections, data fusion methodology, or physical layer signalling details.

22.4 Probabilistic Framework for Distributed Processing

The information being handled in multi-sensor systems almost always relates to a state of nature and, consequently, it is assumed to be unknown prior to observation or estimation. Thus, the model of the information flow shown in Figure 22.2 is probabilistic, and hence can be quantified using the principles of information theory [28,29]. Furthermore, the process of data detection and processing that occurs within the sensors and fusion node(s) can be considered as elements of classical statistical decision theory [30]. Using the mature techniques that these disciplines offer, a probabilistic information-processing relation can be quantified for sensor networks and analyzed within the framework of the well-known Bayesian paradigm [31]. The basic tasks in this approach are the following:

1. Determination of appropriate information-processing techniques, models, and metrics for fusion and sensor administration.
2. Representation of the sensors process, data fusion, and administration methodologies using the appropriate probabilistic models.
3. Analysis of the measurable aspects of the information flow in the sensor architecture using the defined models and metrics.
4. Design of optimal data fusion algorithms and architectures for optimal inference in multi-sensor systems.

© 2005 by Chapman & Hall/CRC

5. Design, implementation, and test of associated networking and physical-layer algorithms and architectures for the models determined in (4).

We now consider two issues in information combining in multi-sensor systems: (i) the nature of the information being generated by the sensors and (ii) the method of combining the information from disparate sources.

22.4.1 Sensor Data Model for Single Sensors

Any observation or measurement by any sensor is always uncertain to a degree determined by the *precision* of the sensor. This uncertainty, or measurement *noise*, requires us to treat the data generated by a sensor probabilistically. We therefore adopt the notation and definitions of probability theory to determine an appropriate model for sensor data [2,34].

Definition 22.1. A *state vector* at time instant t is a representation of the *state of nature* of a process of interest, and can be expressed as a vector $\boldsymbol{x}(t)$ in a measurable, finite-dimensional vector space Ω over a discrete or continuous field $\mathcal{F}$:

$$\boldsymbol{x}(t) \in \Omega \subseteq \mathbb{R}^n \tag{22.1}$$

The state vector is arbitrarily assumed to be n-dimensional and can represent a particular state of nature of interest, e.g. it can be the three-dimensional position vectors of an airplane. The state space may be either continuous or discrete (e.g. the on or off states of a switch).

Definition 22.2. A *measurement vector* at time instant t is the information generated by a single sensor (in response to an observation of nature), and can be represented by an m-dimensional vector $\boldsymbol{z}(t)$ from a measurement vector space Ψ.

$$\boldsymbol{z}(t) = \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{pmatrix} \in \Psi \subseteq \mathbb{R}^m \tag{22.2}$$

Intuitively, the measurement vector may be thought of as m pieces of data that a single sensor generates from a single observation at a single instant of time. Because of measurement error, the sensor output $\boldsymbol{z}(t)$ is an approximation of $\boldsymbol{x}(t)$ — the true state of nature. It is important to note that $\boldsymbol{z}(t)$ may itself not be directly visible to the user of the sensor platform. A noise-corrupted version $\Gamma\{\boldsymbol{z}(t), \boldsymbol{\nu}(t)\}$, as defined below, may be all that is available for processing. Furthermore, the dimensionality of the sensor data may not be the same as the dimension of the observed parameter that is being measured. For example, continuing with the airplane example, a sensor may display the longitude and latitude of the airplane at a particular instant of time via global positioning system (a two-dimensional observation vector), but may not be able to measure the altitude of the airplane (which completes the three-dimensional specification of the actual location of the airplane in space).

The *measurement error* itself can be considered as another vector, $\boldsymbol{\nu}(t)$, or a *noise process* vector, of the same dimensionality as the observation vector $\boldsymbol{z}(t)$. As the name suggests, noise vectors are inherently stochastic in nature, and serve to render all sensor measurements uncertain, to a specific degree.

Definition 22.3. An *observation model* Γ for a sensor is a mapping from state space Ω to observation space Ψ, and is parameterized by the statistics of the noise process:

$$\Gamma_{\boldsymbol{\nu}} : \Omega \mapsto \Psi \tag{22.3}$$

© 2005 by Chapman & Hall/CRC

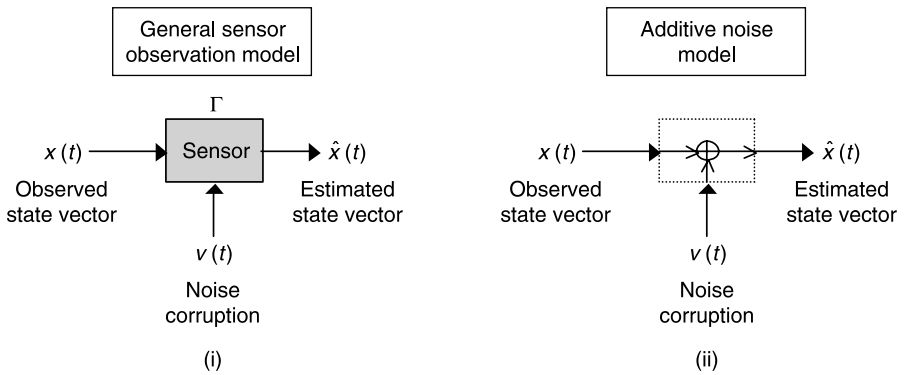


Figure 22.3. Sensor data models: (i) general case; (ii) noise additive case.

Functionally, the relationship between the state, observation, and noise vectors can be expressed as

$$\boldsymbol{z}(t) = \Gamma\{\boldsymbol{x}(t), v(t)\} \tag{22.4}$$

Objective. The objective in sensing applications is to infer the unknown state vector $\boldsymbol{x}(t)$ from the error-corrupted and (possibly lower dimensional) observation vector $\boldsymbol{z}(t), \boldsymbol{v}(t)$. If the functional specification of the mapping in Equation (22.3), and the noise vector $\boldsymbol{v}(t)$, were known for all times t, then finding the inverse mapping for one-to-one cases would be trivial, and the objective would be easily achieved. It is precisely because either or both parameters may be random that gives rise to various estimation architectures for inferring the state vector from the imperfect observations. A geometric interpretation of the objective can be presented, as shown in Figure 22.3(i). The simplest mapping relationship Γ that can be used as a sensor data model is the *additive* model of noise corruption, as shown in Figure 22.3(ii), which can be expressed as

$$\boldsymbol{x} = \Gamma(\boldsymbol{z} + \boldsymbol{v}) \tag{22.5}$$

Typically, for well designed and matched sensor platforms, the noise vector is small compared with the measurement vector, in which case a Taylor approximation can be made:

$$\boldsymbol{x} = \Gamma(\boldsymbol{z}) + (\nabla_z \Gamma)\boldsymbol{z} + (\text{higher order terms}) \tag{22.6}$$

where ∇_z is the Jacobian matrix of the mapping Γ with respect to the state measurement vector $\boldsymbol{z}$. Since the measurement error is random, the state vector observed is also random, and we are in essence dealing with random variables. Thus, we can use well-established statistical methods to quantify the uncertainty in the random variables [31]. For example, the statistics of the noise process $\boldsymbol{v}(t)$ can often be known *a priori*. Moments are the most commonly used measures for this purpose; in particular, if the covariance of the noise process is known, $\mathbf{E}\{\boldsymbol{v}\boldsymbol{v}^{\mathrm{T}}\}$, then the covariance of the state vector is [2]

$$\mathbf{E}\{\boldsymbol{x}\boldsymbol{x}^{\mathrm{T}}\} = (\nabla_z \Gamma)\mathbf{E}\{\boldsymbol{v}\boldsymbol{v}^{\mathrm{T}}\}(\nabla_z \Gamma)^{\mathrm{T}} \tag{22.7}$$

For uncorrelated noise $\boldsymbol{v}$, the matrix $(\nabla_z \Gamma)\mathbf{E}\{\boldsymbol{v}\boldsymbol{v}^{\mathrm{T}}\}(\nabla_z \Gamma)^{\mathrm{T}}$ is symmetric and can be decomposed using singular value decomposition [32]:

$$(\nabla_z \Gamma)\mathbf{E}\{\boldsymbol{v}\boldsymbol{v}^{\mathrm{T}}\}(\nabla_z \Gamma)^{\mathrm{T}} = (\mathrm{SDS}^{\mathrm{T}}) \tag{22.8}$$

© 2005 by Chapman & Hall/CRC

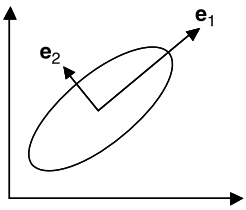


Figure 22.4. Ellipsoid of state vector uncertainty.

where S is an $(n \times n)$ matrix of orthogonal vectors $\boldsymbol{e}_j$ and D are the eigenvalues of the decomposition:

$$\mathsf{S} = (\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_n), \quad \boldsymbol{e}_i \boldsymbol{e}_j = \begin{cases} 1 & \text{for } i = j \\ 0 & \text{for } i \neq j \end{cases} \tag{22.9}$$

$$\mathsf{D} = \text{diag}\,(d_1, d_2, \ldots, d_n) \tag{22.10}$$

The components of D correspond to the scalar variance in each of direction. Geometrically, all the directions for a given state $\boldsymbol{x}$ can be visualized as an ellipsoid in n-dimensional space, with the principal axes in the directions of the vectors $\boldsymbol{e}_k$ and $2\sqrt{d_j}$ as the corresponding magnitudes. The volume of the ellipsoid is the uncertainty in $\boldsymbol{x}$. The two-dimensional case is shown in Figure 22.4. From this perspective, the basic objective in the data fusion problem is then to reduce the volume of the uncertainty ellipsoid. All the techniques for data estimation, fusion, and inference are designed towards this goal [33].

22.4.2 A Bayesian Scheme for Decentralized Data Fusion

Given the inherent uncertainty in measurements of states of nature, the end goal in using sensors, as mentioned in the previous section, is to obtain the best possible estimates of the states of interest for a particular application. The Bayesian approach to solving this problem is concerned with quantifying *likelihoods* of events, given various types of partial knowledge or observations, and subsequently determining the state of nature that is most probably responsible for the observations as the "best" estimate.

The issue of whether the Bayesian approach is intrinsically the "best" approach for a particular problem[2] is a philosophical debate that is not discussed here further. However, it may be mentioned that, arguably, the Bayesian paradigm is most *objective* because it is based only on observations and "impartial" models for sensors and systems.

The information contained in the (noise-corrupted) measured state vector $\boldsymbol{z}$ is first described by means of probability distribution functions (PDFs). Since all observations of states of nature are causal manifestations of the underlying processes governing the state of nature,[3] the PDF of $\boldsymbol{z}$ is conditioned by the state of nature at which time the observation/measurement was made. Thus, the PDF of $\boldsymbol{z}$ conditioned by $\boldsymbol{x}$ is what is usually measurable and is represented by

$$F_Z(\boldsymbol{z} \mid \boldsymbol{x}) \tag{22.11}$$

This is known as the *likelihood function* for the observation vector. Next, if information about the possible states under observation is available (e.g. *a priori* knowledge of the range of possible states),

[2]In contrast with various other types of inferential and subjective approaches [31].

[3]Ignoring the observer–state interaction difficulties posed by Heisenberg uncertainty considerations.

© 2005 by Chapman & Hall/CRC

or more precisely the probability distribution of the possible states $F_X(\boldsymbol{x})$, then the prior information and the likelihood function [Equation (22.11)] can be combined to provide the *a posteriori* conditional distribution of $\boldsymbol{x}$, given $\boldsymbol{z}$, by Bayes' theorem [34]:

Theorem 22.1.

$$F_X(\boldsymbol{x} \mid \boldsymbol{z}) = \frac{F_Z(\boldsymbol{z} \mid \boldsymbol{x})F_X(\boldsymbol{x})}{\int_\Omega F_Z(\boldsymbol{z} \mid \boldsymbol{x})F_X(\boldsymbol{x})\,\mathrm{d}F(\boldsymbol{x})} = \frac{F_Z(\boldsymbol{z} \mid \boldsymbol{x})F_X(\boldsymbol{x})}{F_Z(\boldsymbol{z})} \tag{22.12}$$

Usually, some function of the actual likelihood function, $g(T(\boldsymbol{z}) \mid \boldsymbol{x})$, is commonly available as the processable information from sensors. $T(\boldsymbol{z})$ is known as the *sufficient statistic* for $\boldsymbol{x}$ and Equation (22.12) can be reformulated as

$$F_X(\boldsymbol{x} \mid \boldsymbol{z}) = F_X(\boldsymbol{x} \mid T(\boldsymbol{z})) = \frac{g(T(\boldsymbol{z}) \mid \boldsymbol{x})F_X(\boldsymbol{x})}{\int_\Omega g(T(\boldsymbol{z}) \mid \boldsymbol{x})F_X(\boldsymbol{x})\,\mathrm{d}F(\boldsymbol{x})} \tag{22.13}$$

When observations are carried out in discrete time steps according to a desired resolution, then a vector formulation is possible. Borrowing notation from Manyika and Durrant-Whyte [2], all observations up to time index r can be defined as

$$\boldsymbol{Z}^r \triangleq \{\boldsymbol{z}(1), \boldsymbol{z}(2), \ldots, \boldsymbol{z}(r)\} \tag{22.14}$$

from where the posterior distribution of $\boldsymbol{x}$ given the set of observations $\boldsymbol{Z}^r$ becomes

$$F_X(\boldsymbol{x} \mid \boldsymbol{Z}^r) = \frac{F_{Z^r}(\boldsymbol{Z}^r \mid \boldsymbol{x})F_X(\boldsymbol{x})}{F_{Z^r}(\boldsymbol{Z}^r)} \tag{22.15}$$

Using the same approach, a recursive version of Equation (22.15) can also be formulated:

$$F_X(\boldsymbol{x} \mid \boldsymbol{Z}^r) = \frac{F_Z(\boldsymbol{z}(r) \mid \boldsymbol{x})F_X(\boldsymbol{x} \mid \boldsymbol{Z}^{r-1})}{F_Z(\boldsymbol{z}(r) \mid \boldsymbol{Z}^{r-1})} \tag{22.16}$$

in which case all the r observations do not need to be stored, and instead only the current observation $\boldsymbol{z}(r)$ can be considered at the rth step. This version of Bayes' law is most prevalent in practice, since it offers a directly implementable technique for fusing observed information with *prior beliefs*.

22.4.2.1 Classical Estimation Techniques

A variety of inference techniques can now be applied to estimate the state vector $\boldsymbol{x}$ (from the time series observations from a single sensor). The estimate, denoted by $\hat{\boldsymbol{x}}$, is derived from the posterior distribution $F_{\boldsymbol{x}}(\boldsymbol{x} \mid \boldsymbol{Z}^r)$ and is a point in the uncertainty ellipsoid of Figure 22.4. The basic objective is to reduce the volume of the ellipsoid, which is equivalent to minimizing the probability of error based on some criterion. Three classical techniques are now briefly reviewed: *maximum likelihood*(ML), *maximum a posteriori*(MAP) and *minimum mean-square error*(MMSE) estimation.

ML estimation involves maximizing the likelihood function [Equation (22.11)] by some form of search over the state space Ω:

$$\hat{\boldsymbol{x}}_{\mathrm{ML}} = \arg\max_{\boldsymbol{x} \in \Omega} F_{Z^r}(\boldsymbol{Z}^r \mid \boldsymbol{x}) \tag{22.17}$$

© 2005 by Chapman & Hall/CRC

This is intuitive, since the PDF is greatest when the correct state has been guessed for the conditioning variable. However, a major drawback is that, for state vectors from large state spaces, the search may be computationally expensive or infeasible. Nonetheless, this method is widely used in many disciplines, e.g. digital communication reception [35].

The MAP estimation technique involves maximizing the posterior distribution from observed data as well as from prior knowledge of the state space:

$$\hat{\boldsymbol{x}}_{\mathrm{MAP}} = \arg\max_{\boldsymbol{x}\in\Omega} F_{\boldsymbol{x}}(\boldsymbol{x} \mid \boldsymbol{Z}^r) \tag{22.18}$$

Since prior information may be subjective, objectivity for an estimate (or the inferred state) is maintained by considering only the likelihood function (i.e. only the observed information). In the instance of no prior knowledge, and the state space vectors are all considered to be equally likely, the MAP and ML criteria can be shown to be identical.

MMSE techniques attempt to minimize the estimation error by searching over the state space, albeit in an organized fashion. This is the most popular technique in a wide variety of information-processing applications, since the variable can often be found analytically, or the search space can be reduced considerably or investigated systematically. The key notion is to reduce the covariance of the estimate. Defining the mean and variance of the posterior observation variable as

$$\boldsymbol{x} \triangleq \mathrm{E}_{F(\boldsymbol{x}|\boldsymbol{Z}^r)}\{\boldsymbol{x}\} \tag{22.19}$$

$$\mathrm{Var}(\boldsymbol{x}) \triangleq \mathrm{E}_{F(\boldsymbol{x}|\boldsymbol{Z}^r)}\{(\boldsymbol{x}-\bar{\boldsymbol{x}})(\boldsymbol{x}-\bar{\boldsymbol{x}})^{\mathrm{T}}\} \tag{22.20}$$

it can be shown that the least-squares estimator is one that minimizes the Euclidean distance between the true state $\boldsymbol{x}$ and the estimate $\hat{\boldsymbol{x}}$, given the set of observations $\boldsymbol{Z}^r$. In the context of random variables, this estimator is referred to as the MMSE estimate and can be expressed as

$$\hat{\boldsymbol{x}}_{\mathrm{MMSE}} = \arg\min_{\boldsymbol{x}\in\Omega} \mathrm{E}_{F(\boldsymbol{x}|\boldsymbol{Z}^r)}\{(\boldsymbol{x}-\bar{\boldsymbol{x}})(\boldsymbol{x}-\bar{\boldsymbol{x}})^{\mathrm{T}}\} \tag{22.21}$$

To obtain the minimizing estimate, Equation (22.21) can be differentiated with respect to $\hat{\boldsymbol{x}}$ and set equal to zero, which yields $\hat{\boldsymbol{x}} = \mathrm{E}\{\boldsymbol{x} \mid \boldsymbol{Z}^r\}$. Thus, the MMSE estimate is the conditional mean. It also can be shown that the MMSE estimate is the minimum variance estimate; and when the conditional density coincides with the mode, the MAP and MMSE estimators are equivalent.

These estimation techniques and their derivatives, such as the Wiener and Kalman filters [36], all serve to reduce the uncertainty ellipsoid associated with state $\boldsymbol{x}$ [33]. In fact, direct applications of these mathematical principles formed the field of radio-frequency *signal detection* in noise, and shaped the course of developments in digital communication technologies.

22.4.3 Distributed Detection Theory and Information Theory

Information theory was developed to determine the fundamental limits on the performance of communication systems [37]. Detection theory, on the other hand, involves the application of statistical decision theory to estimate states of nature, as discussed in the previous section. Both these disciplines can be used to treat problems in the transmission and reception of information, as well as the more general problem of data fusion in distributed systems. The synergy was first explored by researchers in the 1950s and 1960s [38], and the well-established source and channel coding theories were spawned as a result. With respect to data fusion, the early research in the fields of information theory and fusion proceeded somewhat independently. Whereas information theory continued exploring the limits of digital signalling, data fusion, on the other hand, and its myriad *ad hoc* techniques were developed by

© 2005 by Chapman & Hall/CRC

the practical concerns of signal detection, aggregation, and interpretation for decision making. Gradually, however, it was recognized that both issues, at their abstract levels, dealt fundamentally with problems of information processing.

Subsequently, attempts were made to unify distributed detection and fusion theory, e.g. as it applied in sensor fusion, with the broader field of information theory. Some pioneering work involved the analysis of the hypothesis testing problem using discrimination [39], employing cost functions based on information theory for optimizing signal detection [38], and formulating the detection problem as a coding problem for asymptotic analysis using error exponent functions [40,41]. More recently, research in these areas has been voluminous, with various theoretical studies exploring the performance limits and asymptotic analysis of fusion and detection schemes [11,42].

In particular, some recent results [3] are relevant to the case of a distributed system of sensor nodes. As has been noted earlier, the optimal engineering trade-offs for the efficient design for such a system are not always clear cut. However, if the detection/fusion problem can be recast in terms of information-theoretic cost functions, then it has been shown that system optimization techniques provide useful design paradigms.

For example consider the block diagrams of a conventional binary detection system and a binary communication channel shown in Figure 22.5. The source in the detection problem can be viewed as the information source in the information transmission problem. The decisions in the detection model can be mapped as the channel outputs in the channel model. Borrowing the notation from Varshney [3], if the input is considered a random variable $H = i,\ i = 0, 1$, where probability $P(H = 0) = P_0$, the output $u = i,\ i = 0, 1$, is then a decision random variable, whose probabilities of detection (P_D), miss (P_M), false alarm (P_F), etc. can be interpreted in terms of the transition probabilities of the information transmission problem. This is the classic example of the binary channel [35].

If the objective of the decision problem is the minimization of the information loss between the input and output, then it can be shown that the objective is equivalent to the maximization of the *mutual information* $I(H; u)$ (see Section 22.5 for formal definitions of entropy and information measures). This provides a mechanism for computing practical likelihood test ratios as a technique for *information-optimal* data fusion. Thus, for the case of the binary detection problem, the *a posteriori* probabilities are:

$$P(u = 0) = P_0(1 - P_F) + (1 - P_0)(1 + P_D) \triangleq \alpha_0 \tag{22.22}$$

$$P(u = 1) = P_0 P_F + (1 - P_0)P_D \triangleq \alpha_1 \tag{22.23}$$

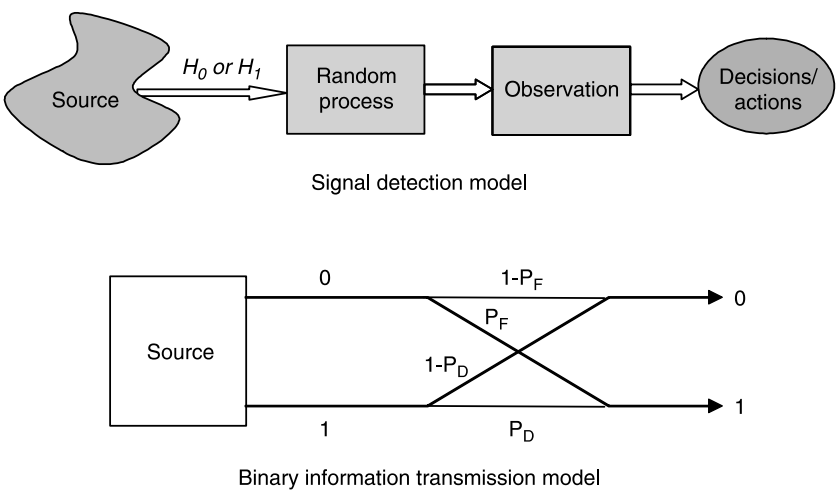


Figure 22.5. Signal detection versus information transmission.

© 2005 by Chapman & Hall/CRC

whereupon it can be shown that the optimal decision threshold for the received signal is

$$\text{Threshold} = \frac{-P_0\{\log(\alpha_0/\alpha_1) - \log[(1-P_F)/P_F]\}}{(1-P_0)\{\log(\alpha_0/\alpha_1) - \log[(1-P_D)/P_D]\}} \tag{22.24}$$

This approach can be extended to the case of distributed detection. For example, for a detection system in a parallel topology without a fusion center, and assuming the observations at the local detectors are conditionally independent, the goal is then to maximize the mutual information $I(H; \boldsymbol{u})$ where the vector $\boldsymbol{u}$ contains the local decisions. Once again, it can be shown that the optimal detectors are threshold detectors, and likelihood ratio tests can then be derived for each detector. Using the second subscript in the variables below to refer to the detector number, the thresholds are

$$\text{Threshold}_1 = -\frac{P_0\left[\log\left(\frac{\alpha_{00}}{\alpha_{10}}\right) + P_{F2}\log\left(\frac{\alpha_{01}\alpha 10}{\alpha_{00}\alpha 11}\right) - \log\left(\frac{1-P_{F1}}{P_{F1}}\right)\right]}{(1-P_0)\left[\log\left(\frac{\alpha_{00}}{\alpha_{10}}\right) + P_{D2}\log\left(\frac{\alpha_{01}\alpha 10}{\alpha_{00}\alpha 11}\right) - \log\left(\frac{1-P_{D1}}{P_{D1}}\right)\right]} \tag{22.25}$$

with a similar expression for Threshold_2. In a similar manner, other entropy-based information-theoretic criteria (e.g. logarithmic cost functions) can be successfully used to design the detection and distributed fusion rules in an integrated manner for various types of fusion architectures (e.g. serial, parallel with fusion center, etc.). This methodology provides an attractive, unified approach for system design, and has the intuitive appeal of treating the distributed detection problem as an information transmission problem.

22.5 Bayesian Framework for Distributed Multi-Sensor Systems

When a number of spatially and functionally different sensor systems are used to observe the same (or similar) state of nature, then the data fusion problem is no longer simply a state-space uncertainty minimization issue. The distributed and multi-dimensional nature of the problem requires a technique for checking the usefulness and validity of the data from each of the not necessarily independent sensors. The data fusion problem is more complex, and general solutions are not readily evident. This section explores some of the commonly studied techniques and proposes a novel, simplified methodology that achieves some measure of generality.

The first issue is the proper modeling of the data sources. If there are p sensors observing the same state vector, but from different vantage points, and each one generates its own observations, then we have a collection of observation vectors $\boldsymbol{z}_1(t), \boldsymbol{z}_2(t), \ldots, \boldsymbol{z}_p(t)$, which can be represented as a combined matrix of all the observations from all sensors (at any particular time t):

$$\boldsymbol{Z}(t) = \big(\boldsymbol{z}_1(t)\ \boldsymbol{z}_2(t)\ \cdots\ \boldsymbol{z}_p(t)\big) = \begin{bmatrix} z_{11} & z_{21} & \cdots & z_{p1} \\ z_{12} & z_{22} & \cdots & z_{p2} \\ & & \ddots & \\ z_{1m} & z_{2m} & \cdots & z_{pm} \end{bmatrix} \tag{22.26}$$

Furthermore, if each sensor makes observations up to time step r for a discretized (sampled) observation scheme, then the matrix of observations $\boldsymbol{Z}(\boldsymbol{r})$ can be used to represent the observations of all the p sensors at time-step r (a discrete variable, rather than the continuous $\boldsymbol{Z}(t)$). With adequate memory allocation for signal processing of the data, we can consider the super-matrix $\{\boldsymbol{Z}^r\}$ of all the observations of all the p sensors from time step 0 to r:

$$\{\boldsymbol{Z}^r\} = \bigcup_{i=1}^{p} \boldsymbol{Z}_i^r \tag{22.27}$$

© 2005 by Chapman & Hall/CRC

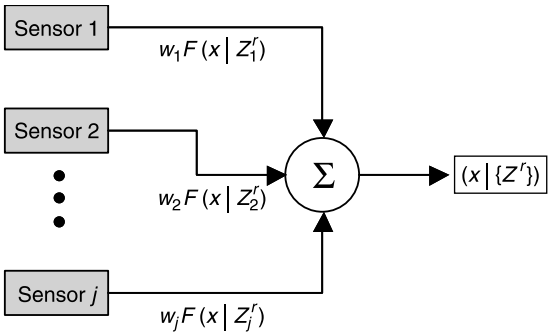


Figure 22.6. Multi-sensor data fusion by linear opinion pool.

where

$$\boldsymbol{Z}_i^r = \left\{\boldsymbol{z}_i(1), \boldsymbol{z}_i(2), \ldots \boldsymbol{z}_i(r)\right\} \tag{22.28}$$

To use all the available information for effectively fusing the data from multiple sensors, this suggests that what is required is the global posterior distribution $F_{\boldsymbol{x}}(\boldsymbol{x} \mid \{\boldsymbol{Z}^r\})$, given the time-series information from each source. This can be accomplished in a variety of ways, the most common of which are summarized below [2].

The *linear opinion pool* [43] aggregates probability distributions by linear combinations of the local posterior PDF information $F_{\boldsymbol{x}}\left(\boldsymbol{x} \mid \boldsymbol{Z}_i^r\right)$ [or appropriate likelihood functions, as per Equation (22.11)]:

$$F(\boldsymbol{x} \mid \{\boldsymbol{Z}^r\}) = \sum_j w_j F\left(\boldsymbol{x} \mid \boldsymbol{Z}_j^r\right) \tag{22.29}$$

where the weights w_j sum to unity and each weight w_j represents a subjective measure of the reliability of the information from sensor j. The process can be illustrated as shown in Figure 22.6. Bayes' theorem can now be applied to Equation (22.29) to obtain a recursive form, which is omitted here for brevity. One of the shortcomings of the linear opinion pool method is its inability to reinforce opinion because the weights are usually unknown except in very specific applications.

The *independent opinion pool* is a product form modification of the linear opinion pool and is defined by the product

$$F(\boldsymbol{x} \mid \{\boldsymbol{Z}^r\}) = \alpha \prod_j F\left(\boldsymbol{x} \mid \boldsymbol{Z}_j^r\right) \tag{22.30}$$

where α is a normalizing constant. The fusion process in this instance can be illustrated as shown in Figure 22.7.

This model is widely used, since it represents the case when the observations from the individual sensors are essentially independent. However, this is also its weakness, since if the data are correlated at a group of nodes, then their opinion is multiplicatively reinforced, which can lead to error propagation in faulty sensor networks. Nevertheless, this technique is appropriate when the prior state-space distributions are truly independent and equally likely (as is common in digital communication applications).

To counter the weaknesses of the two common approaches summarized above, a third fusion rule is the *likelihood opinion pool*, defined by the following recursive rule:

$$F(\boldsymbol{x} \mid \{\boldsymbol{Z}^r\}) = \alpha F\left(\boldsymbol{x} \mid \left\{\boldsymbol{Z}^{r-1}\right\}\right)\left[\prod_j \underbrace{F\left(\boldsymbol{z}_j(r) \mid \boldsymbol{x}\right)}_{\text{likelihood}}\right] \tag{22.31}$$

© 2005 by Chapman & Hall/CRC

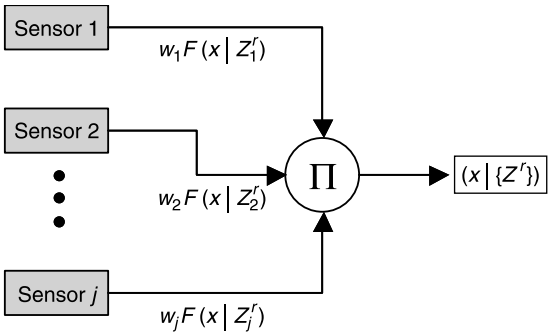


Figure 22.7. Multi-sensor data fusion by independent opinion pool.

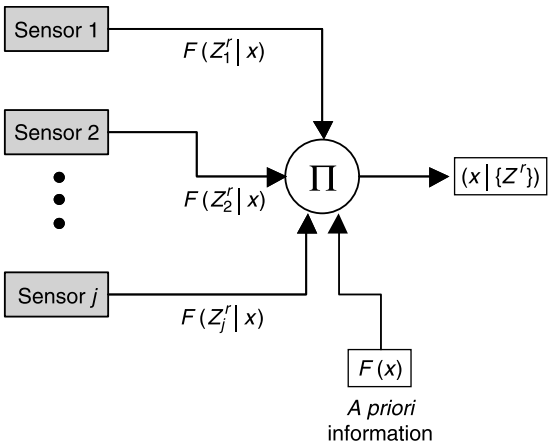


Figure 22.8. Multi-sensor data fusion by likelihood opinion pool.

The likelihood opinion pool method of data fusion can be illustrated as shown in Figure 22.8. The likelihood opinion pool technique is essentially a Bayesian update process and is consistent with the recursive process derived in general in Equation (22.16). It is interesting to note that a simplified, specific form of this type of information processing occurs in the so-called *belief propagation* [44] types of algorithm that are widespread in artificial intelligence and the decoding theory for channel codes. In the exposition above, however, the assumptions and derivations are explicitly identified and derived, and are thus in a general form that is suitable for application to heterogeneous multi-sensor systems. This provides intuitive insight as to how the probabilistic updates help to reinforce "opinions" when performing a distributed state-space search.

22.5.1 Information-Theoretic Justification of the Bayesian Method

Probability distributions allow a quantitative description of the observables, the observer, and associated errors. As such, the likelihood functions and distributions contain information about the underlying states that they describe. This approach can be extended further to actually incorporate measures for the information contained in these random variables. In this manner, an information-theoretic justification can be obtained for the likelihood opinion pool for multi-sensor data fusion, as discussed in the previous section. Some key concepts from information theory [28] are required first.

© 2005 by Chapman & Hall/CRC

22.5.2 Information Measures

The connections between information theory and distributed detection [3] were briefly surveyed in Section 22.3. In this section, some formal information measures are defined to enable an intuitive information-theoretic justification of the utility of the Bayesian update method. This approach also provides an insight towards the practical design of algorithms based on the likelihood opinion pool fusion rules that have been discussed earlier.

To build an information-theoretic foundation for data fusion, the most useful fundamental metric is the Shannon definition of Entropy.

Definition 22.4. *Entropy* is the uncertainty associated with a probability distribution, and is a measure of the descriptive complexity of a PDF [45]. Mathematically:

$$h\{F(\boldsymbol{x})\} \triangleq \mathrm{E}\{-\ln F(\boldsymbol{x})\} \tag{22.32}$$

Note that alternative definitions of the concept of information which predate Shannon's formulation, e.g. the *Fisher information matrix* [46], are also relevant and useful, but not discussed here further.

Using this definition, an expression for the entropy of the posterior distribution of $\boldsymbol{x}$ given $\boldsymbol{Z}^r$ at time r (which is the case of multiple observations from a single sensor) can be expressed as

$$h(r) \triangleq h\left\{F(\boldsymbol{x} \mid \boldsymbol{Z}^r)\right\} = -\sum F(\boldsymbol{x} \mid \boldsymbol{Z}^r) \ln F(\boldsymbol{x} \mid \boldsymbol{Z}^r) \tag{22.33}$$

Now, the entropy relationship for Bayes' theorem can be developed as follows:

$$\begin{aligned} \mathrm{E}\{-\ln[F(\boldsymbol{x} \mid \boldsymbol{Z}^r)]\} &= \mathrm{E}\left\{-\ln[F(\boldsymbol{x} \mid \boldsymbol{Z}^{r-1})]\right\} \\ &\quad - \mathrm{E}\left\{\ln\left[\frac{F(\boldsymbol{z}(r) \mid \boldsymbol{x})}{F(\boldsymbol{z}(r) \mid \boldsymbol{Z}^{r-1})}\right]\right\} \end{aligned} \tag{22.34}$$

$$\Rightarrow h(r) = h(r-1) - \mathrm{E}\left\{\ln\left[\frac{F(\boldsymbol{z}(r) \mid \boldsymbol{x})}{F(\boldsymbol{z}(r) \mid \boldsymbol{Z}^r - 1)}\right]\right\} \tag{22.35}$$

This is an alternative form of the result that conditioning with respect to observations reduces entropy [28]. Using the definition of mutual information, Equation (22.34) can be written in an alternative form as shown below.

Definition 22.5. For an observation process, *mutual information* at time r is the information about $\boldsymbol{x}$ contained in the observation $\boldsymbol{z}(\boldsymbol{r})$:

$$I(x, z(r)) \triangleq \mathrm{E}\left\{\ln\left[\frac{F(\boldsymbol{z}(r) \mid \boldsymbol{x})}{F(\boldsymbol{z}(r))}\right]\right\} \tag{22.36}$$

from where

$$h(r) = h(r-1) - \boldsymbol{I}(r) \tag{22.37}$$

© 2005 by Chapman & Hall/CRC

which means that the entropy following an observation is reduced by an amount equal to the information inherent in the observation.

The insight to be gained here is that, by using the definitions of entropy and mutual information, the recursive Bayes update procedure derived in Equation (22.16) can now be seen as an information update procedure:

$$\mathrm{E}\{\ln[F(\boldsymbol{x} \mid \boldsymbol{Z}^r)]\} = \mathrm{E}\{\ln[F(\boldsymbol{x} \mid \boldsymbol{Z}^{r-1})]\} + \mathrm{E}\left\{\ln\left[\frac{F(\boldsymbol{z}(r) \mid \boldsymbol{x})}{F(\boldsymbol{z}(r) \mid \boldsymbol{Z}^{r-1})}\right]\right\} \tag{22.38}$$

which can be interpreted as [2]

Posterior information = Prior information + Observation information

The information update equation for the likelihood opinion pool fusion rule thus becomes

$$\mathrm{E}\{\ln[F(\boldsymbol{x} \mid \boldsymbol{Z}^r)]\} = \mathrm{E}\{\ln[F(\boldsymbol{x} \mid \boldsymbol{Z}^{r-1})]\} + \sum_j \mathrm{E}\left\{\ln\left[\frac{F(\boldsymbol{z}_j(r) \mid \boldsymbol{x})}{F(\boldsymbol{z}_j(r) \mid \boldsymbol{Z}^{r-1})}\right]\right\} \tag{22.39}$$

The utility of the log-likelihood definition is that the information update steps reduce to simple additions, and are thus amenable to hardware implementation without such problems as overflow and dynamic range scaling.

Thus the Bayesian probabilistic approach is theoretically self-sufficient for providing a unified framework for data fusion in multi-sensor platforms. The information-theoretic connection to the Bayesian update makes the approach intuitive, and shows rigorously how the likelihood opinion pool method serves to reduce the ellipsoid uncertainty. This framework answers the question of how to weight or process outputs of diverse sensors, whether they have different sensing modes of signal-to-noise ratios, without resort to *ad hoc* criteria. Acoustic, visual, magnetic, and other signals can all be combined [47]. Further, since trade-offs in information rate and distortion can be treated using entropies (rate distortion theory [29]), as of course can communication, questions about fundamental limits in sensor networks can now perhaps be systematically explored.

Of course, obvious practical difficulties remain, such as how to determine the uncertainty in measurements, the entropy of sources, and in general how to convert sensor measurements into entropies efficiently.

22.6 Concluding Remarks

In this chapter, the approach of using a probabilistic, information-processing approach to data fusion in multi-sensor networks was discussed. The Bayesian approach was seen to be the central unifying tool in formulating the key concepts and techniques for decentralized organization of information. Thus, it offers an attractive paradigm for implementation in a wide variety of systems and applications. Further, it allows one to use information-theoretic justifications of the fusion algorithms, and also offers preliminary asymptotic analysis of large-scale system performance.

The information-theoretic formulation makes clear how to combine the outputs of possibly entirely different sensors. Moreover, it allows sensing, signal processing, and communication to be viewed in one mathematical framework. This may allow systematic study of many problems involving the cooperative interplay of these elements. This can further lead to the computation of fundamental limits on performance against with practical reduced complexity techniques can be compared.

References

[1] Pottie, G. *et al.*, Wireless sensor networks, in *Information Theory Workshop Proceedings*, Killamey, Ireland, June 22–26, 1998.

© 2005 by Chapman & Hall/CRC

[2] Manyika, J. and Durrant-Whyte, H., *Data Fusion and Sensor Management*, Ellis Horwood Series in Electrical and Electronic Engineering, Ellis Horwood, West Sussex, UK, 1994.
[3] Varshney, P.K., *Distributed Detection and Data Fusion*, Springer-Verlag, New York, NY, 1997.
[4] Popoli, R., The sensor management imperative, in *Multi-Target Multi-Sensor Tracking*, Bar-Shalom, Y. (ed.), Artech House, 325, 1992.
[5] Kuc, R. and Siegel, M.W. Physically based simulation model for acoustic sensor robot navigation, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(6), 766, 1987.
[6] Luo, R. and Kay, M., Multi-sensor integration and fusion in intelligent systems, *IEEE Transactions on Systems Man and Cybernetics*, 19(5), 901, 1989.
[7] Mitchie, A. and Aggarwal, J.K., Multiple sensor integration through image processing: a review, *Optical Engineering*, 23(2), 380, 1986.
[8] Leonard, J.J., Directed sonar sensing for mobile robot navigation, Ph.D. dissertation, University of Oxford, 1991.
[9] Waltz, E. and Llinas, J., *Multi-Sensor Data Fusion*, Artech House, 1991.
[10] Berger, T. *et al.*, Model distribution in decentralized multi-sensor fusion, in *Proceedings of the American Control Conference (ACC)*, 2291, 1991.
[11] Tsitsiklis, J.N., On the complexity of decentralized decision-making and detection problems, *IEEE Transactions on Automatic Control*, 30(5), 440, 1985.
[12] Gamal, E. and Cover, T.M., Multiple user information theory, *Proceedings of the IEEE*, 68, 1466, 1980.
[13] Csiszar, I. and Korner, J., Towards a general theory of source networks, *IEEE Transactions on Information Theory*, IT-26, 155, 1980.
[14] Frank, R., *Understanding Smart Sensors*, Artech House, Norwood, MA, 2000.
[15] Rai-Choudhury, P. (ed.), *MEMS and MOEMS Technology and Applications*, Society of Photo-Optical Instrumentation Engineers, 2000.
[16] Kucar, A.D., Moblie radio — an overview, *IEEE Personal Communications Magazine*, 72, November, 1991.
[17] Royer, E. and Toh, C.-K., A review of current routing protocols for *ad hoc* wireless networks, *IEEE Personal Communications Magazine*, 6(2), 46, 1999.
[18] Sohrabi, K. and Pottie, G., Performance of a self-organizing algorithm for wireless *ad hoc* sensor networks, in *IEEE Vehicular Technology Conference*, Fall, 1999.
[19] Pottie, G., Hierarchical information processing in distributed sensor networks, in *IEEE International Symposium on Information Theory*, Cambridge, MA, August 16–21, 1998.
[20] Durrant-Whyte, H., Sensor models and multi-sensor integration, *International Journal of Robotics*, 7(6), 97, 1988.
[21] Flynn, A.M., Combining ultra-sonic and infra-red sensors for mobile robot navigation, *International Journal of Robotics Research*, 7(5), 5, 1988.
[22] Garvey, T. *et al.*, Model distribution in decentralized multi-sensor fusion, in *Proceedings of the American Control Conference*, 2291, 1991.
[23] Verdu, S., *Multiuser Detection*, Cambridge University Press, 1998.
[24] Balchen J. *et al.*, Structural solution of highly redundant sensing in robotic systems, in *Highly Redundant Sensing in Robotic Systems*, NATO Advanced Science Institutes Series, vol. 58, Springer-Verlag, 1991.
[25] Sohrabi, K. *et al.*, Protocols for self-organization for a wireless sensor network, *IEEE Personal Communication Magazine*, 6, October, 2000.
[26] Singh, S. *et al.*, Power-aware routing in mobile ad hoc networks, in *Proceedings of the 4th Annual IEEE/ACM International Conference on Mobile Computing and Networking (MOBICOM)*, Dallas, TX, 181, 1998.
[27] Yao, K. *et al.*, Blind beamforming on a randomly distributed sensor array, *IEEE Journal on Selected Areas in Communication*, 16(8), 1555, 1998.

© 2005 by Chapman & Hall/CRC

[28] Cover, T.M. and Thomas, J.A., *Elements of Information Theory*, Wiley-Interscience, Hoboken, NJ, 1991.
[29] Gallager, R.G., *Information Theory and Reliable Communications*, John Wiley & Sons, New York, NY, 1968.
[30] Poor, H.V., *An Introduction to Signal Detection and Estimation*, Springer-Verlag, New York, NY, 1988.
[31] Roussas, G.G., *A Course in Mathematical Statistics*, 2nd ed., Harcourt/Academic Press, Burlington, MA, 1997.
[32] Scheick, J.T., *Linear Algebra with Applications*, McGraw-Hill, New York, NY, 1996.
[33] Nakamura, Y., Geometric fusion: minimizing uncertainty ellipsoid volumes, *Data Fusion, Robotics and Machine Intelligence*, Academic Press, 1992.
[34] Fristedt, B. and Gray, L., *A Modern Approach to Probability Theory. Probability and its Applications*, Boston, MA, Birkhauser, 1997.
[35] Proakis, J.G., *Digital Communications*, McGraw-Hill, New York, NY, 2000.
[36] Kalman, R.E., A new approach to linear filtering and prediction problems, *Transactions of the ASME Journal of Basic Engineering*, 82(D), 34, 1969.
[37] Shannon, C.E., A mathematical theory of communication, *Bell Systems Technical Journal*, 27, 279, 1948.
[38] Middleton, D., *Statistical Communication Theory*, McGraw-Hill, 1960.
[39] Kullback, S., *Information Theory and Statistics*, John Wiley & Sons, New York, NY, 1959.
[40] Csiszar, I. and Longo, G., On the error exponent for source coding and for testing simple statistical hypotheses, *Studia Scientiarum Mathematicarum Hungarica.*, 6, 181, 1971.
[41] Blahut, R.E., Hypothesis testing and information theory, *IEEE Transactions on Information Theory*, 20(4), 405, 1974.
[42] Blum, R.S. and Kassam, S.A., On the asymptotic relative efficiency of distributed detection schemes, *IEEE Transactions on Information Theory*, 41(2), 523, 1995.
[43] Stone, M., The opinion pool, *The Annals of Statistics*, 32, 1339, 1961.
[44] Pearl, J., *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann, 1997.
[45] Catlin, D., *Estimation, Control and the Discrete Kalman Filter*, Springer-Verlag, 1989.
[46] Fisher, R.A., On the mathematical foundations of theoretical statistics, *Philosophical Transactions of the Royal Society of London, series A*, 222, 309, 1922.
[47] John, W.F. III. *et al.*, Statistical and information-theoretic methods for self-organization and fusion of multimodal, networked sensors, *International Journal of High Performance Computing Applications*, 2001.

© 2005 by Chapman & Hall/CRC

23

Multispectral Sensing

N.K. Bose

23.1 Motivation

Sensing is ubiquitous in multitudinous applications that include biosensing, chemical sensing, surface acoustic wave sensing, sensing coupled with actuation in control, and imaging sensors. This chapter will be concerned primarily with multispectral sensing that spans acquisition, processing, and classification of data from multiple images of the same scene at several spectral regions. The topic of concern here finds usage in surveillance, health care, urban planning, ecological monitoring, geophysical exploration, and agricultural assessment. The field of sensing has experienced a remakable period of progress that necessitated the launching of an IEEE journal devoted exclusively to that topic in June 2001.

Sensors, photographic or nonphotographic, are required in data acquisition prior to processing and transmission. Light and other forms of electromagnetic (EM) radiation are commonly described in terms of their wavelengths (or freqencies), and the sensed data may be in different portions of the EM spectrum. Spectroscopy is the study of EM radiation as a function of wavelength that has been emitted, reflected, or scattered from a solid, liquid, or gas. The complex interaction of light with matter involves reflection and refraction at boundaries of materials, a process called scattering, and absorption by the medium as light passes through the medium. Scattering makes reflectance spectroscopy possible. The amount of light scattered and absorbed by a grain is dependent on the grain size. Reflectance spectroscopy can be used to map exposed minerals from aircraft, including detailed clay mineralogy. Visual and near-infrared spectroscopy, on the other hand, is insensitive to some minerals that do not have absorptions in this wavelength region. For a comprehensive survey, the reader is referred to [1].

Photographic film is limited for use in the region from near ultraviolet (wavelength range: 0.315 to 0.380 μm) to near infrared (wavelength range: 0.780 to 3 μm). Electronic sensors (like radar, scanners, photoconductive or tube sensors) and solid-state sensors [like charge coupled device (CCD) arrays], though more complicated and bulkier than comparable photographic sensors, are usable over a wider frequency range, in diurnal as well as nocturnal conditions, and are more impervious to fog, clouds, pollution and bad weather. Infrared sensors are indispensable under nocturnal and limited visibility conditions, while electro-optic sensing systems using both absorption lidar and infrared spectroscopy

© 2005 by Chapman & Hall/CRC

have been widely used in both active and passive sensing of industrial and atmospheric pollutants, detection of concealed explosives for airport security applications, detection of land mines, and weather monitoring through the sensing and tracking of vapor clouds. A 40-year review of the infrared imaging system modeling activities of the U.S. Army Night Vision and Electronic Sensor Directorate (NVESD) is available in the inaugural issue of the *IEEE Sensors Journal* [2].

A vast majority of image sensors today are equipped with wavelength-sensitive optical filters that produce multispectral images which are characterized as locally correlated but globally independent random processes. For monochrome and color television, solid-state sensors are being increasingly preferred over photoconductive sensors because of greater compactness and well-defined structure inspite of the fact that solid-state sensors have lower signal-to-noise ratio and lower spatial resolution. The disadvantages, owing to physical constraints like the number of sensor elements that can be integrated on a chip, are presently being overcome through the development of an innovative technical device that gave birth to superresolution imaging technology.

23.2 Introduction to Multispectral Sensing

A significant advance in sensor technology stemmed from the subdividing of spectral ranges of radiation into bands. This allowed sensors in several bands to form multispectral images [3]. From the time that Landsat 1 was launched in 1972, multispectral sensing has found diverse uses in terrain mapping, agriculture, material identification, and surveillance. Typically, multispectral sensors collect several separate spectral bands with spectral regions selected to highlight particular spectral characteristics. The number of bands range from one (panchromatic sensors) to, progressively, tens, hundreds, and thousands of narrow adjacent bands in the case of multispectral, hyperspectral, and ultraspectral sensing respectively. The *spectral resolution* of a system for remote sensing depends on the number and widths (spectral bandwidths) of the spectral bands collected.

Reflectance is the percentage of incident light that is reflected by a material and the reflectance spectrum shows the reflectance of a material across a range of wavelengths, which can often permit unique identification of the material. Many terrestrial minerals have very unique spectral signatures, like human fingerprints, due to the uniqueness of their crystal geometries. Multispectral sensing is in multiple, separated, and narrow wavelength bands. Hyperspectral sensors, on the other hand, operate over wider contiguous bands. Multispectral sensors can usually be of help in detecting, classifying and, possibly, distinguishing between materials, but hyperspectral sensors may actually be required to characterize and identify the materials. Ultraspectral is beyond hyperspectral, with a goal of accommodating, ultimately, millions of very narrow bands for a truly high-resolution spectrometer that may be capable of quantifying and predicting. The need for ultraspectral sensing and imaging is because of the current thrust in chemical, biological and nuclear warfare monitoring, quantification of ecological pollutants, gaseous emission and nuclear storage monitoring, and improved crop assessments though weed identification and prevention.

23.2.1 Instruments for Multispectral Data Acquisition

In remote sensing, often the input imagery is obtained from satellite sensors like Landsat multispectral scanners (MSS) or airborne scanners, synthetic aperture radars (for obtaining high-resolution imagery at microwave frequencies), infrared photographic film, image tubes, and optical scanners (for infrared images) and electro-optical line scanners in addition to the commonly used photographic and television devices (for capturing the visible spectrum) [4]. Multispectral instruments image the Earth in a few strategic areas of the EM spectrum, omitting entire wavelength sections. *Spatial resolution* is the smallest ground area that can be discerned in an image. In Landsat images (nonthermal bands), the spatial resolution is about 28.5 m $\times$ 28.5 m. The smallest discernible area on the ground is called the *resolution*

© 2005 by Chapman & Hall/CRC

cell and determines the sensor's maximum resolution. For a homogeneous feature to be detected, its size, generally, has to be equal to or larger than the resolution cell. *Spectral resolution* is the smallest band or portion of the EM spectrum in which objects are discernible. This resolution defines the ability of a sensor to define wavelength intervals. *Temporal resolution* is the shortest period of time in which a satellite will revisit a spot on the Earth's surface. Landsat 5, for example, has a temporal resolution of 16 days. *Radiometric resolution* is the smallest size of a band or portion of the EM spectrum in which the reflectance of a feature may be assigned a digital number, i.e. the finest distinction that can be made between objects in the same part of the EM spectrum. It describes the imaging system's ability to discriminate between very slight differences in energy. Greater than 12 bits and less than 6 bits correspond, respectively, to very high and low radiometric resolutions. Continuous improvements in spatial, spectral, radiometric, and temporal resolution, coupled with decreasing cost, are making remote sensing techniques very popular.

A scanning system used to collect data over a variety of different wavelengths is called an MSS. MSS systems have several advantages over conventional aerial photographic systems, including the following:

- The ability to capture data from a wider portion of the EM spectrum (about 0.3 to 14 μm).
- The ability to collect data from multiple spectral bands simultaneously.
- The data collected can be transmitted to Earth to avoid storage problems.
- The data collected are easier to calibrate and rectify.

23.2.2 Array and Super-Array Sensing

Acquisition of multivariate information from the environment often requires extensive use of sensing arrays. A localized miniature sensing array can significantly improve the sensing performance, and deployment of a large number of such arrays as a distributed sensing network (super-array) will be required to obtain high-quality information from the environment. Development of super-arrays is a trend in many fields, like the chemical field, where the environment could be gaseous. Such arrays could provide higher selectivity, lower thresholds of detection, broader dynamic range, and long-term baseline stability. Array sensors have been used in a variety of applications that are not of direct interest here. It suffices to single out two such potential areas. An array of plasma-deposited organic film-coated quartz crystal resonators has been studied for use in indoor air-monitoring in aircraft cabins, automobiles, trains or clean rooms [5]. Multiple sensors are also capable of carrying out remote sewer inspection tasks, where closed-circuit television-based platforms are less effective for detecting a large proportion of all possible damages because of the low quality of the acquired images [6]. Multisensors and the superresolution technology, discussed in subsequent sections, are therefore very powerful tools for solving challenging problems in military, civil, and health-care problems.

23.2.3 Multisensor Array Technology for Superresolution

Multiple, undersampled images of a scene are often obtained by using a CCD detector array of sensors which are shifted relative to each other by subpixel displacements. This geometry of sensors, where each sensor has a subarray of sensing elements of suitable size, has recently been popular in the task of attaining spatial resolution enhancement from the acquired low-resolution degraded images that comprise the set of observations. Multisensor array technology is particularly suited to microelectromechanical systems applications, where accuracy, reliability, and low transducer failure rates are essential in applications spanning chronic implantable sensors, monitoring of semiconductor processes, mass-flow sensors, optical cross-connect switches, and pressure and temperature sensors. The benefits include application to any sensor array or cluster, reduced calibration and periodic

© 2005 by Chapman & Hall/CRC

maintenance costs, higher confidence in sensor measurements based on statistical average on multiple sensors, extended life of the array compared with a single-sensor system, improved fault tolerance, lower failure rates, and low measurement drift. Owing to hardware cost, size, and fabrication complexity limitations, imaging systems like CCD detector arrays often provide only multiple low-resolution degraded images. However, a high-resolution image is indispensable in applications such as health diagnosis and monitoring, military surveillance, and terrain mapping by remote sensing. Other intriguing possibilities include substituting expensive high-resolution instruments like scanning electron microscopes by their cruder, cheaper counterparts and then applying technical methods for increasing the resolution to that derivable with much more costly equipment. Small perturbations around the ideal subpixel locations of the sensing elements (responsible for capturing the sequence of undersampled degraded frames), because of imperfections in fabrication, limit the performance of the signal-processing algorithms for processing and integrating the acquired images for the desired enhanced resolution and quality. Resolution improvement by applying tools from digital signal-processing techniques has, therefore, been a topic of very great interest.

23.3 Mathematical Model for Multisensor Array-Based Superresolution

A very fertile arena for applications of some of the developed theory of multidimensional systems has been spatio-temporal processing following image acquisition by, say a single camera, mutiple cameras, or an array of sensors. An image acquisition system composed of an array of sensors, where each sensor has a subarray of sensing elements of suitable size, has recently been popular for increasing the spatial resolution with high signal-to-noise ratio beyond the performance bound of technologies that constrain the manufacture of imaging devices.

A brief introduction to a mathematical model used in high-resolution image reconstruction is provided first. Details can be found in Bose and Boo [7]. Consider a sensor array with $L_1 \times L_2$ sensors in which each sensor has $N_1 \times N_2$ sensing elements (pixels) and the size of each sensing element is $T_1 \times T_2$. The goal is to reconstruct an image of resolution $M_1 \times M_2$, where $M_1 = L_1 N_1$ and $M_2 = L_2 N_2$ To maintain the aspect ratio of the reconstructed image, the case where $L_1 = L_2 = L$ is considered. For simplicity, L is assumed to be an even positive integer in the following discussion.

To generate enough information to resolve the high-resolution image, subpixel displacements between sensors are necessary. In the ideal case, the sensors are shifted from each other by a value proportional to $T_1/L \times T_2/L$. However, in practice there can be small perturbations around these ideal subpixel locations due to imperfections of the mechanical imaging system during fabrication. Thus, for $l_1, l_2 = 0, 1, \ldots, L-1$, with $(l_1, l_2) \neq (0, 0)$, the horizontal and vertical displacements $d^x_{l_1 l_2}$ and $d^y_{l_1 l_2}$ respectively of the $[l_1, l_2]$-th sensor with respect to the $[0, 0]$-th reference sensor are given by

$$d^x_{l_1 l_2} = \frac{T_1}{L}(l_1 + \bar{\epsilon}^x_{l_1 l_2}) \quad \text{and} \quad d^y_{l_1 l_2} = \frac{T_2}{L}(l_2 + \bar{\epsilon}^y_{l_1 l_2})$$

where $\bar{\epsilon}^x_{l_1 l_2}$ and $\bar{\epsilon}^y_{l_1 l_2}$ denote, respectively, the actual normalized horizontal and vertical displacement errors. The estimates of these parameters, $\epsilon^x_{l_1 l_2}$ and $\epsilon^y_{l_1 l_2}$, can be obtained by manufacturers during camera calibration. It is reasonable to assume that

$$|\bar{\epsilon}^x_{l_1 l_2}| < \frac{1}{2} \quad \text{and} \quad |\bar{\epsilon}^y_{l_1 l_2}| < \frac{1}{2}$$

© 2005 by Chapman & Hall/CRC

because if that is not the case, then the low-resolution images acquired from two different sensors may have more than the desirable overlapping information for reconstructing satisfactorily the high-resolution image [7].

Let $f(x_1, x_2)$ denote the original bandlimited high-resolution scene, as a function of the continuous spatial variables x_1, x_2. Then, the observed low-resolution digital image $\bar{g}_{l_1 l_2}$ acquired from the (l_1, l_2)-th sensor, characterized by a point-spread function, is modeled by

$$\bar{g}_{l_1 l_2}[n_1, n_2] = \int_{T_2(n_2 - \frac{1}{2}) + d^y_{l_1 l_2}}^{T_2(n_2 + \frac{1}{2}) + d^y_{l_1 l_2}} \int_{T_1(n_1 - \frac{1}{2}) + d^x_{l_1 l_2}}^{T_1(n_1 + \frac{1}{2}) + d^x_{l_1 l_2}} f(x_1, x_2)\, \mathrm{d}x_1 \mathrm{d}x_2 \tag{23.1}$$

for $n_1 = 1, \ldots, N_1$ and $n_2 = 1, \ldots, N_2$. These low-resolution images are combined to yield the $M_1 \times M_2$ high-resolution image $\bar{g}$ by assigning its pixel values according to

$$\bar{g}[L(n_1 - 1) + l_1, L(n_2 - 1) + l_2] = \bar{g}_{l_1 l_2}[n_1, n_2] \tag{23.2}$$

for $l_1, l_2 = 0, 1, \ldots, (L-1)$, $n_1 = 1, \ldots, N_1$ and $n_2 = 1, \ldots, N_2$.

The continuous image model $f(x_1, x_2)$ in Equation (23.1) can be discretized by the rectangular rule and approximated by a discrete image model. Let $\mathbf{g}$ and $\mathbf{f}$ be, respectively, the vectors formed from discretization of $g(x_1, x_2)$ and $f(x_1, x_2)$ using a column ordering. The Neumann boundary condition [8] is applied on the images. This assumes that the scene immediately outside is a reflection of the original scene at the boundary, i.e.

$$f(i,j) = f(k,l) \quad \text{where} \quad \begin{cases} k = 1 - i & i < 1 \\ k = 2M_1 + 1 - i & i > M_1 \\ l = 1 - j & j < 1 \\ l = 2M_2 + 1 - j & j > M_2 \end{cases}$$

Under the Neumann boundary condition, the blurring matrices are banded matrices with bandwidth $L+1$, but the entries at the upper left part and the lower right part of the matrices are changed. The resulting matrices, denoted by $\mathbf{H}^x_{l_1 l_2}(\bar{\epsilon}^x_{l_1, l_2})$ and $\mathbf{H}^y_{l_1 l_2}(\bar{\epsilon}^y_{l_1, l_2})$, each have a Toeplitz-plus-Hankel structure, as shown in Equation (23.4). The blurring matrix corresponding to the (l_1, l_2)-th sensor under the Neumann boundary condition is given by the Kronecker product:

$$\mathbf{H}_{l_1 l_2}(\bar{\epsilon}_{l_1, l_2}) = \mathbf{H}^x_{l_1 l_2}(\bar{\epsilon}^x_{l_1, l_2}) \otimes \mathbf{H}^y_{l_1 l_2}(\bar{\epsilon}^y_{l_1, l_2})$$

where the 2×1 vector $\bar{\epsilon}_{l_1, l_2}$ is denoted by $(\bar{\epsilon}^x_{l_1, l_2}\ \bar{\epsilon}^y_{l_1, l_2})^{\mathrm{T}}$. The blurring matrix for the whole sensor array is made up of blurring matrices from each sensor:

$$\mathbf{H}_L(\bar{\epsilon}) = \sum_{l_1=0}^{L-1} \sum_{l_2=0}^{L-1} \mathbf{D}_{l_1 l_2} \mathbf{H}_{l_1 l_2}(\bar{\epsilon}_{l_1, l_2}) \tag{23.3}$$

where the $2L^2 \times 1$ vector $\bar{\epsilon}$ is defined as $\bar{\epsilon} = [\bar{\epsilon}^x_{00}\ \bar{\epsilon}^y_{00}\ \bar{\epsilon}^x_{01}\ \bar{\epsilon}^y_{01}\ \cdots\ \bar{\epsilon}^x_{L-1L-2}\ \bar{\epsilon}^y_{L-1L-2}\ \bar{\epsilon}^x_{L-1L-1}\ \bar{\epsilon}^y_{L-1L-1}]^{\mathrm{T}}$. Here $\mathbf{D}_{l_1 l_2}$ are diagonal matrices with diagonal elements equal to 1 if the corresponding component of $\mathbf{g}$ comes from the (l_1, l_2)-th sensor and zero otherwise. The Toeplitz-plus-Hankel matrix $\mathbf{H}^x_{l_1 l_2}(\bar{\epsilon}^x_{l_1, l_2})$,

© 2005 by Chapman & Hall/CRC

referred to above is explicitly written next.

$$\mathbf{H}^x_{l_1l_2}(\bar{\epsilon}^x_{l_1,l_2}) = \frac{1}{L}\begin{pmatrix} \overbrace{1 & \cdots & 1}^{L/2 \text{ ones}} & \frac{1}{2}-\bar{\epsilon}^x_{l_1l_2} & & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \\ 1 & \ddots & \ddots & \ddots & \ddots & \frac{1}{2}-\bar{\epsilon}^x_{l_1l_2} \\ \frac{1}{2}+\bar{\epsilon}^x_{l_1l_2} & \ddots & \ddots & \ddots & \ddots & 1 \\ & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & & \frac{1}{2}+\bar{\epsilon}^x_{l_1l_2} & \underbrace{1 & \cdots & 1}_{L/2 \text{ ones}} \end{pmatrix}$$

$$+\frac{1}{L}\begin{pmatrix} \overbrace{1 & \cdots & 1}^{L/2-1 \text{ ones}} & \frac{1}{2}+\bar{\epsilon}^x_{l_1l_2} & & 0 \\ \vdots & \iddots & \iddots & & & \\ 1 & \iddots & & & & \frac{1}{2}-\bar{\epsilon}^x_{l_1l_2} \\ \frac{1}{2}+\bar{\epsilon}^x_{l_1l_2} & & & & \iddots & 1 \\ & & & \iddots & \iddots & \vdots \\ 0 & & \frac{1}{2}-\bar{\epsilon}^x_{l_1l_2} & \underbrace{1 & \cdots & 1}_{L/2-1 \text{ ones}} \end{pmatrix}. \tag{23.4}$$

The matrix $\mathbf{H}^y_{l_1l_2}(\bar{\epsilon}^y_{l_1,l_2})$ is defined similarly. See Bose and Boo [7] for more details.

23.3.1 Image Reconstruction Formulation

CCD image sensor arrays, where each sensor consists of a rectangular subarray of sensing elements, produce discrete images whose sampling rate and resolution are determined by the physical size of the sensing elements. If multiple CCD image sensor arrays are shifted relative to each other by exact subpixel values, then the reconstruction of high-resolution images can be modeled by

$$\bar{g} = Hf \quad \text{and} \quad g = \bar{g} + \eta \tag{23.5}$$

where f is the desired high-resolution image, H is the blur operator, $\bar{g}$ is the output high-resolution image formed from low-resolution frames and η is the additive Gaussian noise. However, as perfect subpixel displacements are practically impossible to realize, blur operators in multisensor high-resolution image reconstruction are space variant.

Since the system described in Equation (23.5) is ill-conditioned, solution for f is constructed by applying the maximum *a posteriori* (MAP) regularization technique. This involves a functional $R(f)$, which measures the regularity of f, and a regularization parameter α that controls the degree of regularity of the solution to the minimization problem:

$$\min_f \left\{ \|Hf - g\|_2^2 + \alpha R(f) \right\} \tag{23.6}$$

© 2005 by Chapman & Hall/CRC

The boundary values of g are not completely determined by the original image f inside the scene, because of the blurring process. They are also affected by the values of f outside the scene. Therefore, when solving for f from Equation (23.5), one needs some assumptions on the values of f outside the scene, referred to as boundary conditions. Bose and Boo [7] imposed zero boundary condition outside the scene. Ng and Yip [8] recently showed that the model with the Neumann boundary condition gives a better reconstructed high-resolution image than obtainable with the zero boundary condition. In the case of Neumann boundary conditions, discrete cosine transform (DCT)-based preconditioners have been effective in the high-resolution reconstruction problem [8]. Ng and Bose [9] provides an analysis and proof of convergence of the iterative method deployed to solve the transform-based preconditioned system. The proof offered of linear convergence of the conjugate gradient method on the displacement errors caused from the imperfect locations of subpixels in the sensor array fabrication process has also been substantiated by results from simulation.

The observed signal vector $\bar{g}$ is, as seen above, subject to errors. It is assumed that the actual signal $g = [g_1 \ldots g_{M_1 M_2}]^{\mathrm{T}}$ can be represented by

$$\bar{g} = g + \delta \mathbf{g} \tag{23.7}$$

where

$$\delta g = [\delta g_1 \ \delta g_2 \ \ldots, \ \delta g_{M_1 M_2}]^{\mathrm{T}}$$

and δg_i are independent identically distributed noise with zero-mean and variance σ_g^2.

Thus, the image reconstruction problem is to recover the vector g from the given inexact point spread function $h_{l_1 l_2}$ ($l_1 = 0, 1, \ldots, L_1 - 1$, $l_2 = 0, 1, \ldots, L_2 - 1$) and an observed and noisy signal $\bar{g}$. A constrained total least-squares approach to solving the image resconstruction problem has been advanced [32].

23.3.2 Other Approaches to Superresolution

Multiple undersampled images of a scene are often obtained by using multiple identical image sensors which are shifted relative to each other by subpixel displacements [11,12]. The resulting high-resolution image reconstruction problem using a set of currently available image sensors is interesting, because it is closely related to the design of high-definition television and very high-definition image sensors.

Limitations of image sensing lead to the formation of sequences of undersampled, blurred, and noisy images. High-resolution image reconstruction algorithms, which increase the effective sampling rate and bandwidth of observed low-resolution degraded images, usually accompany a series of processing, tasks, such as subpixel motion estimation, interpolation, and image restoration for tasks in surveillance, medical, and commercial applications. Considerable progress has been made since 1990, when a method based on the recursive least-squares estimation algorithm in the wavenumber domain was proposed to implement simultaneously the tasks of interpolation and filtering of a still image sequence [13]. First, the total least-squares recursive algorithm was developed to generalize the results in [13] to the situation often encountered in practice, when not only the observation is noise corrupted but also the data [14]. The latter scenario originates from the inaccuracies in estimation of the displacements between frames. Second, it was shown that four image sensors are often sufficient from the standpoint of the human visual system to satisfactorily deconvolve moderately degraded multispectral images [15]. Third, it was shown how a three-dimensional (3-D) linear minimum mean-squares error (LMMSE) estimator for a sequence of time-varying images can be decorrelated into a set of two-dimensional (2-D) LMMSE equations that can subsequently be solved by approximating the Karhunen Loeve transform (KLT) by other transforms, like the Hadamard transform or the DCT [16]. Fourth, as already discussed,

© 2005 by Chapman & Hall/CRC

the mathematical model of shifted undersampled images with subpixel displacement errors was derived in the presence of blur and noise, and the MAP formulation was adapted for fast high-resolution reconstruction in the presence of subpixel displacement errors [7]. For discursive documentation of an image acquisition system composed of an array of sensors followed by iterative methods for high-resolution reconstruction and scopes for further research, see Ng and Bose [9,17].

A different approach towards superresolution from that of Kim *et al.* [13] was suggested in 1991 by Irani and Peleg [18], who used a rigid model instead of a translational model in the image registration process and then applied the iterative back-projection technique from computer-aided tomography. A summary of this and other research during the last decade is contained in a recent paper [19]. Mann and Picard [20] proposed the projective model in image registration because their images were acquired with a video camera. The projective model was subsequently used by Lertrattanapanich and Bose [21] for videomosaicing and high resolution. Very recently, an approach towards superresolution using spatial tesselations has been presented [22]. Analysis from the wavelet point of view of the construction of a high-resolution image from low-resolution images acquired through a multisensor array in the approach of Bose and Boo [7] was recently conducted by Chan *et al.* [23]. Absence of displacement errors in the low-resolution samples was assumed, and this resulted in a spatially invariant blurring operator. The algorithms developed decomposed the function from the previous iteration into different wavenumber components in the wavelet transform domain and, subsequently, added them into the new iterate to improve the approximation. Extension of the approach when some of the low-resolultion images are missing, possibly due to sensor failure, was also implemented [23]. The wavelet approach towards high-resolution image formation was generalized to the case of spatially varying blur associated with the presence of subpixel displacement errors due to improper alignment of the sensors [24].

23.4 Color Images

Multispectral restoration of a single image is a 3-D reconstruction problem, where the third axis incorporates different wavelengths. We are interested in color images because there are many applications. Color plays an important role in pattern recognition and digital multimedia, where color-based features and color segmentation have proven pertinent in detecting and classifying objects in satellite and general purpose imagery. In particular, the fusion of color and edge-based features has improved the performance of image segmentation and object recognition.

A color image can be regarded as a set of three images in their primary color channels (red, green and blue). Monochrome processing algorithms applied to each channel independently are not optimal because they fail to incorporate the spectral correlation between the channels. Under the assumption that the spatial intrachannel and spectral interchannel correlation functions are product-separable, Hunt and Kübler [25] showed that a multispectral (e.g. color) image can be decorrelated by the KLT. After decorrelating multispectral images, the Wiener filter can be applied independently to each channel, and the inverse KLT gives the restored color image. In the literature, Galatsanos and Chin [26] proposed and developed the 3-D Wiener filter for processing multispectral images. The 3-D Wiener filter is implemented by using the 2-D block-circulant-circulant-block approximation to a block-Toeplitz-Toepliz-block matrix [27]. Moreover, Tekalp and Pavlovic [28] considered the use of 3-D Kalman filtering with the multispectral image restoration problem.

The visual quality can always be improved by using multiple sensors with distinct transfer characteristics [29]. Boo and Bose [15] developed a procedure to restore a single color image, which has been degraded by a linear shift-invariant blur in the presence of additive noise. Only four sensors, namely red (R), green (G), blur (B) and luminance (Y), are used in the color image restoration problem. In the NTSC YIQ representation, the restoration of the Y component is critical because this component contains 85–95% of the total energy and has a large bandwidth. Two observed luminance images are used to restore the Y component. In their method, the 3-D Wiener filter on a sequence of these two luminance-component images and two 2-D Wiener filters on each of the chrominance-component images are considered. Boo and Bose [15] used circulant approximations in the 2-D or 3-D

© 2005 by Chapman & Hall/CRC

Wiener filters and, therefore, the computational cost can be reduced significantly. The resulting well-conditioned problem is shown to provide improved restoration over the decorrelated component and the independent channel restoration methods, each of which uses one sensor for each of the three primary color components.

Ng and Bose [30] formulated the color image restoration problem with only four sensors (R, G, B, Y) by using the NTSC YIQ decorrelated component method and the Neumann boundary condition, i.e. the data outside the domain of consideration are a reflection of the data inside, in the color image restoration process. Boo and Bose [15] used the traditional choice of imposing the periodic boundary condition outside the scene, i.e. data outside the domain of consideration are exact copies of data inside. The most important advantage of using a periodic boundary condition is that circulant approximations can be used and, therefore, fast Fourier transforms can be employed in the computations. Note that, when this assumption is not satisfied by the images, ringing effects will occur at the boundary of the restored images, e.g. see Boo and Bose [15: figure 6]. The Neumann image model gives better restored color images than that under the periodic boundary condition.

Besides the issue of boundary conditions, it is well known that the color image restoration problem is very ill-conditioned, and restoration algorithms will be extremely sensitive to noise. Ng and Bose [30] used regularized least-squares filters [31,32] to alleviate the restoration problem. It is shown that the resulting regularized least-squares problem can be solved efficiently by using DCTs. In the regularized least-squares formulation, regularization parameters are introduced to control the degree of bias of the solution. The generalized cross-validation function is also used to obtain estimates of these regularization parameters, and then to restore high-quality color images. Numerical examples are given by Ng and Bose [30] to illustrate the effectiveness of the proposed methods over other restoration methods.

Ng *et al.* [33] extended the high-resolution image reconstruction method to multiple undersampled color images. The key issue is to employ the cross-channel regularization matrix to capture the changes of reflectivity across the channels.

23.5 Conclusions

With the increasing need for higher resolution multispectral imagery (MSI) in military and civilian applications, it is felt that the needs of the future can be best addressed by combining the task of deployment of systems with larger collection capability with the technical developments during the last decade in the area of superresolution, briefly summarized in this chapter. The need for model accuracy is undeniable in the attainment of superresolution, along with the design of the algorithm, whose robust implementation will produce the desired quality in the presence of model parameter uncertainty. Since the large volume of collected multispectral data might have to be transmitted prior to the deployment of superresolution algorithms, it is important to attend to schemes for multispectral data compression. Fortunately, the need for attention to MSI compression technology was anticipated about a decade back [34], and the suitability of second-generation wavelets (over and above the standard first generation wavelets) for multispectral (and, possibly hyperspectral and ultraspectral) image coding remains to be investigated.

The optimization problem from regularized least-squares was formulated from the red, green, and blue components (from the RGB sensor) by Ng and Bose [30]. Space-invariant regularization based on generalized cross-validation (GCV) was used. There is considerable scope for incorporating recent regularization methods (like space-variant regularization [35]) in this approach for further improvement in quality of restoration, both in the case of a single image and for multispectral video sequences.

A regularized structured total least-squares algorithm was proposed by Fu and Barlow [36] to perform the image restoration and estimations of the subpixel displacement errors at the same time, unlike the alternate minimization algorithm of Ng *et al.* [10]. The work of Fu and Barlow [36] is based on the use of space-invariant regularization. With the oversimplification of the nature of a real image, however, space-invariant algorithms produce unwanted effects, such as smoothing of sharp edges,

© 2005 by Chapman & Hall/CRC

"ringing" in the vicinity of edges, and noise enhancement in smooth areas of the image [37]. To overcome these unwanted artifacts, many types of space-variant image restoration algorithm have been proposed. Reeves and Mersereau [35] reported the use of an iterative image restoration technique using space-variant regularization. And multichannel restoration of single-channel images using wavelet-based subband decomposition was proposed by Banham and Katsaggelos [37]. Also, image restoration using the subband or wavelet-based approach has been reported by Charbonnier *et al.* [38], and more recently by Chan *et al.* [22,23].

In surveillance systems of tomorrow, signals generated by multiple sensors need to be processed, transmitted, and presented at multiple levels in order to capture the different aspects of the monitored environment. Multiple-level representations to exploit perception augmentation for humans interacting with such systems are also needed in civilian applications to facilitate infrastructure development and urban planning, especially because of the percieved gap in the knowledge base that planners representing different modes of transportation, vegetation, etc. have of each other's constraints or requirements. If both panchromatic and multispectral (or hyperspectral and ultraspectral) images are available, improved automated fusion strategies are needed for the fused image to display sharp features from the panchromatic image while preserving the spectral attributes like color from the multispectral, hyperspectral and ultraspectral image. Situation-awareness techniques that utilize multisensor inputs can provide enhanced indexing capabilities needed for focusing human or robot, fixed or mobile, attention on information of interest. Multiterminal mobile and cooperative alarm detection in surveillance, industrial pollution monitoring, and chemical and biological weapon sensing is another emerging problem where multisensor signal/video data acquisition, compression, transmission, and processing approaches become more and more relevant.

Acknowledgment

This research was supported by ARO grant DAAD 19-03-1-0261.

References

[1] Clark, R.N., Spectroscopy of rocks and minerals, and principles of spectroscopy, in *Manual of Remote Sensing*, vol. 3, Remote Sensing for the Earth Sciences, Rencz, A.N. (ed.), John Wiley, New York, 1999, chap. 1.

[2] Ratches, J.A. *et al.*, Target acquisition performance modeling of infrared imaging systems, *IEEE Sensors Journal*, 1, 31, 2001.

[3] Landgrebe, D.A., *Signal Theory Methods in Multispectral Remote Sensing*, John Wiley, Hoboken, NJ, 2003.

[4] Hord, R.M., *Digital Image Processing of Remotely Sensed Data*, Academic Press, New York, NY, 1982.

[5] Seyama, M. *et al.*, Application of an array sensor based on plasma-deposited organic film coated quartz crystal resonators to monitoring indoor volatile compounds, *IEEE Sensors Journal*, 1, 422, 2001.

[6] Duran, O. *et al.*, State of the art in sensor technologies for sewer inspection, *IEEE Sensors Journal*, 2, 73, 2002.

[7] Bose, N.K. and Boo, K.J., High-resolution image-reconstruction with multisensors, *International Journal of Imaging Systems and Technology*, 9, 294, 1998.

[8] Ng, M. and Yip, A., A fast MAP algorithm for high-resolution image reconstruction with multisensors, *Multidimensional Systems and Signal Processing*, 12(2), 143, 2001.

[9] Ng, M. and Bose, N.K., Analysis of displacement errors in high-resolution image reconstruction with multisensors, *IEEE Transactions on Circuits and Systems, Part I*, 49, 806, 2002.

© 2005 by Chapman & Hall/CRC

[10] Ng, M. *et al.*, Constrained total least squares computations for high resolution image reconstruction with multisensors, *International Journal of Imaging Systems and Technology*, 12, 35, 2002.
[11] Komatsu, T. *et al.*, Signal-processing based method for acquiring very high resolution images with multiple cameras and its theoretical analysis, *Proceedings of IEEE, Part I*, 140(3), 19, 1993.
[12] Jacquemod, G. *et al.*, Image resolution enhancement using subpixel camera displacement, *Signal Processing*, 26, 139, 1992.
[13] Kim, S.P. *et al.*, Recursive reconstruction of high-resolution image from noisy undersampled multiframes, *IEEE Transactions on Acoustics, Speech and Signal Processing*, 38(6), 1013, 1990.
[14] Bose, N.K. *et al.*, Recursive total least squares algorithm for image reconstruction from nosiy undersampled frames, *Multidimensional Systems and Signal Processing*, 4(3), 253, 1993.
[15] Boo, K.J. and Bose, N.K., Multispectral image restoration with multisensors, *IEEE Transactions on Geoscience and Remote Sensing*, 35(5), 1160, 1997.
[16] Boo, K.J. and Bose, N.K., A motion-compensated spatio-temporal filter for image sequences with signal-dependent noise, *IEEE Transactions on Circuits and Systems for Video Tech.*, 8(3), 287, 1998.
[17] Ng, M. and Bose, N.K., Mathematical analysis of super-resolution methodology, *IEEE Signal Processing Magazine*, 20(3), 62, 2003.
[18] Irani, M. and Peleg, S., Improving resolution by image registration, *CVGIP: Graphical Models and Image Processing*, 53, 231, 1991.
[19] Elad, M. and Hel-Or, Y., A fast superresolution reconstruction algorithm for pure translational motion and common space-invariant blur, *IEEE Transactions on Image Processing*, 10, 1187, 2001.
[20] Mann, S. and Picard, R.W., Video orbits of the projective group: a simple approach to featureless estimation of parameters, *IEEE Transactions on Image Processing*, 6, 1281, 1997.
[21] Lertrattanapanich, S. and Bose, N.K., Latest results on high-resolution reconstruction from video sequences, Technical Report of IEICE, DSP 99-140, The Institute of Electronic, Information and Communication Engineers, Japan, December 1999, 59.
[22] Lertrattanapanich, S. and Bose, N.K., High resolution image formation from low resolution frames using Delaunay triangulation, *IEEE Transactions on Image Processing*, 17, 1427, 2002.
[23] Chan, R.F. *et al.*, Wavelet algorithms for high resolution image reconstruction, *SIAM Journal of Scientific Computing*, 24, 1408, 2003.
[24] Chan, R.F. *et al.*, Wavelet deblurring algorithms for spatially varying blur from high resolution image reconstruction, *Linear Algebra and its Applications*, 366, 139, 2003.
[25] Hunt, B. and Kübler, O., Karhunen–Loeve multispectral image restoration, part I: theory, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 32, 592, 1984.
[26] Galatsanos, N. and Chin, R., Digital restoration of multichannel images, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37, 415, 1989.
[27] Bose, N.K. and Boo, K.J., Asymptotic eigenvalue distribution of block-Toeplitz matrices, *IEEE Transactions on Information Theory*, 44(2), 858, 1998.
[28] Tekalp, A. and Pavlovic, G., Multichannel image modeling and Kalman filtering for multispectral image restoration, *Signal Processing*, 19, 221, 1990.
[29] Berenstein, C. and Patrick, E., Exact deconvolution for multiple convolution operators — an overview, plus performance characterization for imaging sensors, *Proceedings of IEEE*, 78, 723, 1990.
[30] Ng, M. and Bose, N.K., Fast color image restoration with multisensors, *International Journal of Imaging Systems and Technology*, 12(5), 189, 2003.
[31] Galatsanos, N. *et al.*, Least squares restoration of multichannel images, *IEEE Transactions on Signal Processing*, 39, 2222, 1991.
[32] Ng, M. and Kwan, W., Comments on least squares restoration of multichannel images, *IEEE Transactions on Signal Processing*, 49, 2885, 2001.
[33] Ng, M. *et al.*, Constrained total least squares for color image reconstruction, *In Total Least Squares and Errors-in-Variables Modelling III: Analysis, Algorithms and Applications*, Huffel, S. and Lemmerling, P. (eds), Kluwer Academic Publishers, 2002, 365.

© 2005 by Chapman & Hall/CRC

[34] Vaughan, V.D. and Atkinson, T.S., System considerations for multispectral image compression designs, *IEEE Signal Processing Magazine*, 12(1), 19, 1995.
[35] Reeves, S.J. and Mersereau, R.M., Optimal estimation of the regularization parameter and stabilizing functional for regularized image restoration, *Optical Engineering*, 29(5), 446, 1990.
[36] Fu, H. and Barlow, J., A regularized structured total least squares algorithm for high resolution image reconstruction, *Linear Algebra and its Applications*, to appear.
[37] Banham, M.R. and Katsaggelos, A.K., Digital image restoration, *IEEE Signal Processing Magazine*, 14(2), 24, 1997.
[38] Charbonnier, P. *et al.*, Noisy image restoration using multiresolution Markov random fields, *Journal of Visual Communication and Image Representation*, 3, 338, 1992.

© 2005 by Chapman & Hall/CRC

IV

Sensor Deployment and Networking

24. Coverage-Oriented Sensor Deployment *Yi Zou and Krishnendu Chakrabarty* *453*
Introduction • Sensor Detection Model • Virtual Force Algorithm for Sensor Node Deployment • Uncertainty Modeling in Sensor Node Deployment • Conclusions

25. Deployment of Sensors: An Overview *S.S. Iyengar, Ankit Tandon, Qishi Wu, Eungchun Cho, Nageswara S.V. Rao, and Vijay K. Vaishnavi* *483*
Introduction • Importance of Sensor Deployment • Placement of Sensors in a DSN using Eisenstein Integers • Complexity Analysis of Efficient Placement of Sensors on Planar Grid • Acknowledgment

26. Genetic Algorithm for Mobile Agent Routing in Distributed Sensor Networks *Qishi Wu, S.S. Iyengar, and Nageswara S.V. Rao* *505*
Introduction • Computational Technique Based on GAs • The MARP • Genetic Algorithm for the MARP • Simulation Results and Algorithm Analysis • Conclusions • Acknowlegment • Appendix A

27. Computer Network — Basic Principles *Suresh Rai* *527*
Introduction • Layered Architecture and Network Components • Link Sharing: Multiplexing and Switching • Data Transmission Basics • Wireless Networks • WLANs • Acknowledgments

28. Location-Centric Networking in Distributed Sensor Networks *Kuang-Ching Wang and Parameswaran Ramanathan* *555*

© 2005 by Chapman & Hall/CRC

Introduction • Location-Centric Computing • Network Model • Location-Centric Networking • Target Tracking Application • Testbed Evaluation

29. **Directed Diffusion** *Fabio Silva, John Heidemann, Ramesh Govindan, and Deborah Estrin* 573
Introduction • Programming a Sensor Network • Directed Diffusion Protocol Family • Facilitating In-Network Processing • Evaluation • Related Work • Conclusion • Acknowledgments

30. **Data Security Perspectives** *David W. Carman* 597
Introduction • Threats • Security Requirements • Constraints • Architecting a Solution • Security Mechanisms • Other Sources • Summary

31. **Quality of Service Metrics** *N. Gautam* 613
Service Systems • QoS in Networking • Systems Approach to QoS Provisioning • Case Studies • Concluding Remarks

32. **Network Daemons for Distributed Sensor Networks** *S.V. Rao and Qishi Wu* 629
Introduction • Network Daemons • Daemons for Wide-Area Networks • Daemons for ***Ad Hoc*** Mobile Networks • Conclusions • Acknowledgments

Till, this book has concentrated primarily on processing sensor data. We have not forgotten that distributed sensor networks (DSNs) are computer networks. This section considers two important issues: how to deploy the networks and how communications is maintained among the nodes. These issues are interconnected. A hostile environment can occlude sensors and/or make communications impossible. In this section, most communications discussions assume a wireless substrate. Both issues also require monitoring node energy expenditures.

Zou and Chakrabarty consider how best to place sensors in order to monitor events in a region. They describe a virtual force algorithm that allows nodes to position themselves in a globally desirable pattern using only local information. In doing so, they introduce many self-organization concepts that will be expanded in Section 7.

Wu *et al.* consider data routing in sensor networks using mobile agents. They phrase routing as an optimization problem. This problem is then solved using genetic algorithms.

Iyengar *et al.* than provide an in-depth analysis of the sensor deployment problem. They use algebraic approaches to consider the efficiency of different tessellation methods. Different methods of describing sensor detection ranges are presented and it is shown that finding the optimal placement of sensors is an NP-complete problem. Again, genetic algorithms are used to tackle this optimization problem.

Rai provides a computer-networking tutorial. This tutorial thoroughly illustrates communications concepts that are used throughout this book. The concepts of protocol layering and data transmission are described in detail. An introduction to wireless communications issues is provided as well.

Ramanathan discusses location-centric networking. In this approach, the network is separated into distinct regions and manger nodes are assigned to coordinate work within the geographic region.

Silva *et al.* explain the concepts behind diffusion routing. Diffusion routing is a technology that has become strongly identified with sensor networking. It is a data-centric communications technology. The implementation described in this chapter uses a publish–subscribe paradigm that changes the way sensor network applications are designed. (The editors can personally attest to this.) This chapter describes both, how the approach is used and its internal design.

Carman discusses data security issues is sensor networks. The chapter starts by describing possible attacks on sensor networks, and the data security requirements of the systems. What makes these

© 2005 by Chapman & Hall/CRC

networks unique, from a data-security perspective, are the numerous operational constraints that must be maintained. A security architecture is then proposed that fulfills the systems' needs without violating the strict resource constraints.

Gautam then presents a tutorial of network Quality of Service (QoS). A network needs to be able to fulfill its demands with a reasonable certainity within time constraints. This chapter discusses how this can be quantified and measured. This type of analysis is essential for distributed systems designs.

Rao and Wu conclude this section by discussing their netlets concept. This concept uses small agile processes to overcome many potential network problems. Network daemons are distributed processes that form an overlay network. They cooperate to overcome many potential network contention problems and provide a more predictable substrate.

This section has considered DSNs as distributed processes. Many networking technologies have been discussed in tutorial fashion. We have discussed how to position the nodes in detail. Network security has been explored, and finally a number of innovative networking technologies presented.

© 2005 by Chapman & Hall/CRC

24

Coverage-Oriented Sensor Deployment

Yi Zou and Krishnendu Chakrabarty

24.1 Introduction

Wireless sensor networks that are capable of observing the environment, processing data, and making decisions based on these observations have recently attracted considerable attention [1–4]. These networks are important for a number of applications, such as coordinated target detection and localization, surveillance, and environmental monitoring. Breakthroughs in miniaturization, hardware design techniques, and system software have led to cheaper sensors and fueled recent advances in wireless sensor networks [1,2,5].

In this chapter, we are focusing on coverage-driven sensor deployment. The coverage of a sensor network refers to the extent to which events in the monitored region can be detected by the sensors deployed. We present strategies for enhancing the coverage of sensor networks with low computation cost, a small number of sensors, and low energy consumption. We also present a probabilistic framework for uncertainty-aware sensor deployment, with applications to air-dropped sensors and deployment through dispersal.

Sensor node deployment problems have been studied in a variety of contexts. In the area of adaptive beacon placement and spatial localization, a number of techniques have been proposed for both fine-grained and coarse-grained localization [6,7]. Sensor deployment and sensor planning for military applications are described by Pottie and Kaiser [3], where a general sensor model is used to detect elusive targets in the battlefield. The sensor coverage analysis is based on a hypothesis of possible target movements and sensor attributes. However, the proposed wireless sensor networks framework by Pottie and Kaiser [3] requires a considerable amount of *a priori* knowledge about possible targets. A variant of sensor deployment has been considered for multi-robot exploration [9,10]. Each robot can be viewed as a sensor node in such systems. An incremental deployment algorithm is used in which sensor nodes are deployed one by one in an adaptive fashion. Each new deployment of a sensor is based on the sensed information from sensors deployed earlier. A drawback of this approach is that it is computationally expensive. As the number of sensors increases, each new deployment results in a relatively large amount of computation.

© 2005 by Chapman & Hall/CRC

The concept of potential force is used by Heo and Varshney [11] in a distributed fashion to perform sensor node deployment in *ad hoc* wireless sensor networks. The problem of evaluating the coverage provided by a given placement of sensors is discussed by Meguerdichian and co-workers [12,13]. The major concern here is the self-localization of sensor nodes; sensor nodes are considered to be highly mobile and they move frequently. An optimal polynomial-time algorithm that uses graph theory and computational geometry constructs is used to determine the best-case and the worst-case coverage. Radar and sonar coverage also present several related challenges. Radar and sonar netting optimization are of great importance for detection and tracking in a surveillance area. Based on the measured radar cross-sections and the coverage diagrams for the different radars, a method has been proposed for optimally locating the radars to achieve satisfactory surveillance with limited radar resources.

Sensor placement on two- and three-dimensional grids has been formulated as a combinatorial optimization problem, and solved using integer linear programming [14,15]. This approach suffers from two main drawbacks. First, computational complexity makes the approach infeasible for large problem instances. Second, the grid coverage approach relies on ''perfect'' sensor detection, i.e. a sensor is expected to yield a binary yes/no detection outcome in every case. However, because of the inherent uncertainty associated with sensor readings, sensor detection must be modeled probabilistically [16,17]. It is well known that there is inherent uncertainty associated with sensor readings; hence, sensor detections must be modeled probabilistically. A probabilistic optimization framework for minimizing the number of sensors for a two-dimensional grid has been proposed recently [16,17]. This algorithm attempts to maximize the average coverage of the grid points.

There also exists a close resemblance between the sensor placement problem and the art gallery problem (AGP) addressed by the art gallery theorem [18]. The AGP can be informally stated as that of determining the minimum number of guards required to cover the interior of an art gallery. (The interior of the art gallery is represented by a polygon.) The AGP has been solved optimally in two dimensions and shown to be NP-hard in the three-dimensional case. Several variants of the AGP have been studied in the literature, including mobile guards, exterior visibility, and polygons with holes.

A related problem in wireless sensor networks is that of spatial localization [7]. In wireless sensor networks, nodes need to be able to locate themselves in various environments and on different distance scales. Localization is particularly important when sensors are not deployed deterministically, e.g. when sensors are thrown from airplanes in a battlefield and for underwater sensors that might move due to drift. Sensor networks also make use of spatial information for self-organization and configuration. A number of techniques for both fine- and coarse-grained localization have been proposed [6,19].

Other related work includes the placement of a given number of sensors to reduce communication cost [20] and optimal sensor placement for a given target distribution [21]. Sensor deployment for collaborative target detection is discussed by Clouqueur *et al.* [22], where path exposure is used as a measure of the effectiveness of the sensor deployment. This method uses sequential deployment of sensors, i.e. a limited number of sensors are deployed in each step until the desired minimum exposure or probability of detection of a target is achieved. In most practical applications, however, we need to deploy the sensors in advance without any prior knowledge of the target, and sequential deployment is often infeasible. Moreover, sequential deployment may be undesirable when the number of sensors or the area of the sensor field is large. Thus, a single-step deployment scheme is more advantageous in such-scenarios. Liu *et al.* [23] propose a dual-space approach to event tracking and sensor resource management.

24.1.1 Chapter Outline

We present a virtual force algorithm (VFA) as a sensor deployment strategy to enhance the coverage after an initial random placement of sensors. The VFA is based on disk packing theory [24] and the virtual force field concept from physics and robotics [9,10]. For a given number of sensors, the VFA attempts to maximize the sensor field coverage. A judicious combination of attractive and repulsive

© 2005 by Chapman & Hall/CRC

forces is used to determine the new sensor locations that improve the coverage. Once the effective sensor positions are identified, a one-time movement with energy consideration incorporated is carried out, i.e. the sensors are redeployed to these positions. The sensor field is represented by a two-dimensional grid. The dimensions of the grid provide a measure of the sensor field. The granularity of the grid, i.e. the distance between grid points, can be adjusted to trade off computation time of the VFA with the effectiveness of the coverage measure. The detection by each sensor is modeled as a circle on the two-dimensional grid, where the center of the circle denotes the sensor and the radius denotes the detection range of the sensor. We first consider a binary detection model in which a target is detected (not detected) with complete certainty by the sensor if a target is inside (outside) its circle. The binary model facilitates the understanding of the VFA model. We then investigate realistic probabilistic models in which the probability that the sensor detects a target depends on the relative position of the target within the circle.

We also formulate an uncertainty-aware sensor deployment problem to model scenarios, where sensor locations are precomputed but the sensors are airdropped or dispersed. In such scenarios, sensor nodes cannot be expected to fall exactly at predetermined locations; rather, there are regions where there is a high probability of a sensor actually being located. Such examples include airdropped sensor nodes and underwater sensor nodes that drift due to water currents. Thus, a key challenge in sensor deployment is to determine an uncertainty-aware sensor field architecture that reduces cost and provides high coverage, even though the exact location of the sensors may not be completely controllable. In this proposal, we present two algorithms for sensor deployment wherein we assumed that sensor positions are not exactly predetermined. We assume that the sensor locations are calculated before deployment and that an attempt is made during the airdrop to place sensors at these locations; however, the sensor placement calculations and coverage optimization are based on a Gaussian model, which assumes that if a sensor is intended for a specific point P in the sensor field, then its exact location can be anywhere in a "cloud" surrounding P. Note that the placement algorithms give us the sensor positions prior to actual placement and we assume that sensors are deployed in a single step.

24.2 Sensor Detection Model

The sensor field is represented by a two-dimensional grid. The dimensions of the grid provide a measure of the sensor field. The granularity of the grid, i.e. distance between grid points, can be adjusted to trade off computation time of the VFA with the effectiveness of the coverage measure. The detection by each sensor is modeled as a circle on the two-dimensional grid. The center of the circle denotes the sensor and the radius denotes the detection range of the sensor. We first consider a binary detection model in which a target is detected (not detected) with complete certainty by the sensor if a target is inside (outside) its circle. The binary model facilitates the understanding of the VFA model. We then investigate two types of realistic probabilistic model in which the probability that the sensor detects a target depends on the relative position of the target.

Let us consider a sensor field represented by an $m \times n$ grid. Let s be an individual sensor node on the sensor field located at grid point (x, y). Each sensor node has a detection range of r. For any grid point P at (i, j), we denote the Euclidean distance between s at (x, y) and P at (i, j) as $d_{ij}(x, y)$, i.e. $d_{ij}(x, y) = \sqrt{(x-i)^2 + (y-j)^2}$. Equation (24.1) shows the binary sensor model [14] that expresses the coverage $c_ij(x, y)$ of a grid point at (i, j) by sensor s at (x, y).

$$c_{ij}(x, y) = \begin{cases} 1 & \text{if } d_{ij}(x,\ y) < r \\ 0 & \text{otherwise} \end{cases} \tag{24.1}$$

The binary sensor model assumes that sensor readings have no associated uncertainty. In reality, sensor detections are imprecise; hence, the coverage $c_{ij}(x, y)$ needs to be expressed in probabilistic terms. A possible way of expressing this uncertainty is to assume the detection probability on a target by a sensor varies exponentially with the distance between the target and the sensor [16,17].

© 2005 by Chapman & Hall/CRC

This probabilistic sensor detection model given in Equation (24.2):

$$c_{ij}(x, y) = e^{-\alpha d_{ij}(x,y)} \tag{24.2}$$

This is also the coverage confidence level of this point from sensor s. The parameter α can be used to model the quality of the sensor and the rate at which its detection probability diminishes with distance. Clearly, the detection probability is unity if the target location and the sensor location coincide. Alternatively, we can also use another probabilistic sensor detection model given in Equation (24.3), which is motivated in part by Elfes [25]:

$$c_{ij}(x,y) = \begin{cases} 0 & \text{if } r + r_e \le d_{ij}(x,y) \\ e^{-\lambda a^{\beta}} & \text{if } r - r_e < d_{ij}(x,y) < r + r_e \\ 1 & \text{if } r - r_e \ge d_{ij}(x,y) \end{cases} \tag{24.3}$$

where r_e ($r_e < r$) is a measure of the uncertainty in sensor detection, $a = d_{ij}(x, y) - (r - r_e)$, and λ and β are parameters that measure detection probability when a target is at distance greater than r_e but within a distance from the sensor. This model reflects the behavior of range-sensing devices, such as infrared and ultrasound sensors. The probabilistic sensor detection model is shown in Figure 24.1. Note that distances are measured in units of grid points. Figure 24.1 also illustrates the translation of a distance response from a sensor to the confidence level as a probability value about this sensor response. Different values of the parameters α and β yield different translations reflected by different detection probabilities, which can be viewed as the characteristics of various types of physical sensor.

It is often the case that there are obstacles in the sensor field terrain. If we are provided with such a priori knowledge about where obstacles in the sensor field and then we can also build the terrain

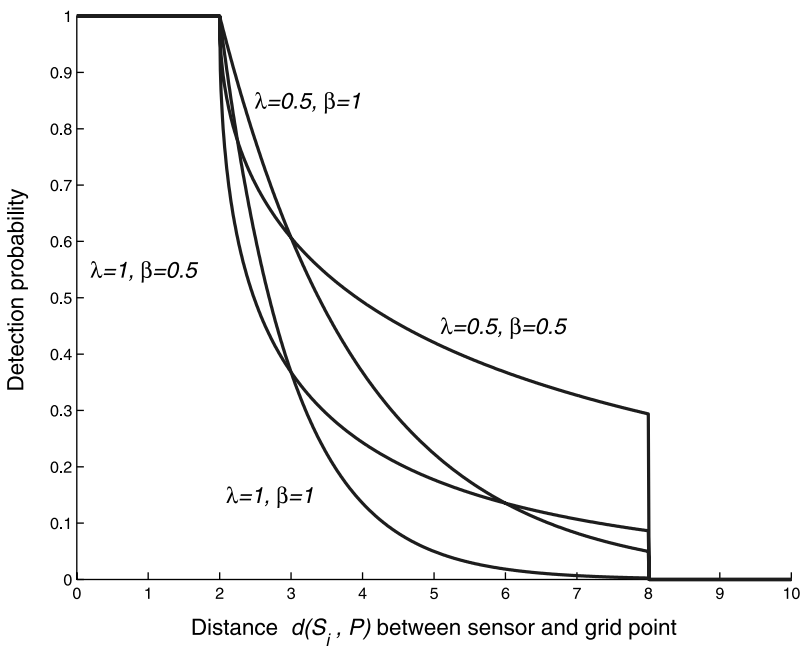


Figure 24.1. Probabilistic sensor detection model.

© 2005 by Chapman & Hall/CRC

information into our models based on the principle of line of sight. An example is given in Figure 24.2. Some types of sensor are not able to see through any obstacles located in the sensor field; hence, models and algorithms must consider the problem of achieving an adequate sensor field coverage in the presence of obstacles. Suppose C_{xy} is an $m \times n$ matrix that corresponds to the detection probabilities of each grid point in the sensor field when a sensor node is located at grid point (x, y), i.e. $C_{xy} = [c_{ij}(x, y)]_{m \times n}$. To achieve the coverage in the presence of obstacles, we need to generate a mask matrix for the corresponding coverage probability matrix C_{xy} to mask out those grid points as the "blocked area," as shown in Figure 24.2. In this way, the sensor node placed at the location (x, y) will not see any grid points beyond the obstacles. We also assume that sensor nodes are not placed on any grid points with obstacles. Figure 24.3 is an example of the mask matrix for a sensor node at $(1, 1)$ in a 10×10 sensor field grid with obstacles located at $(7, 3), (7, 4), (3, 5), (4, 5), (5, 5)$.

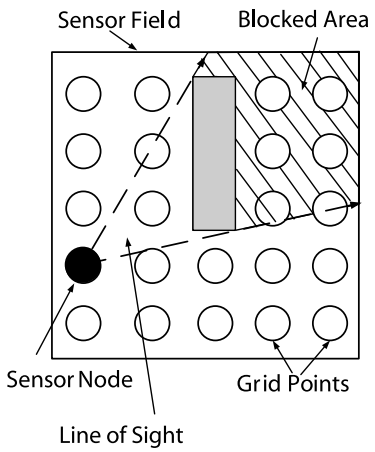


Figure 24.2. Example to illustrate the line-of-sight principle.

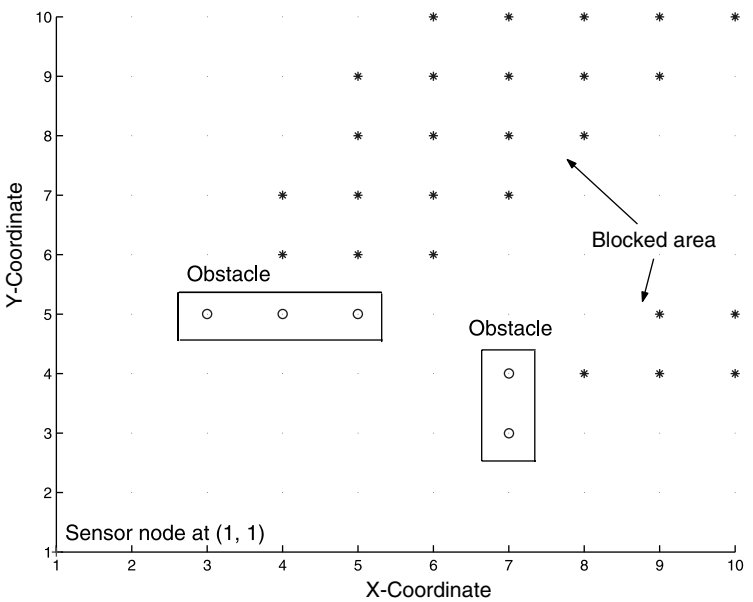


Figure 24.3. Obstacle mask matrix example.

© 2005 by Chapman & Hall/CRC

24.3 Virtual Force Algorithm for Sensor Node Deployment

As an initial sensor node deployment step, a random placement of sensors in the target area (sensor field) is often desirable, especially if no *a priori* knowledge of the terrain is available. Random deployment is also practical in military applications, where wireless sensor networks are initially established by dropping or throwing sensors into the sensor field. However, random deployment does not always lead to effective coverage, especially if the sensors are overly clustered and there is a small concentration of sensors in certain parts of the sensor field. However, the coverage provided by a random deployment can be improved using a force-directed algorithm. We present the VFA as a sensor deployment strategy to enhance the coverage after an initial random placement of sensors. the VFA combines the ideas of potential field [9,10] and disk packing [24]. For a given number of sensors, the VFA attempts to maximize the sensor field coverage using a combination of attractive and repulsive forces. During the execution of the force-directed VFA the sensors do not physically move; rather, a sequence of virtual motion paths is determined for the randomly placed sensors. Once the effective sensor positions are identified, a one-time movement is carried out to redeploy the sensors at these positions. Energy constraints are also included in the sensor repositioning algorithm. In the sensor field, each sensor behaves as a "source of force" for all other sensors. This force can be either positive (attractive) or negative (repulsive). If two sensors are placed too close to each other, with the "closeness" being measured by a predetermined threshold, then they exert negative forces on each other. This ensures that the sensors are not overly clustered, leading to poor coverage in other parts of the sensor field. On the other hand, if a pair of sensors is too far apart from each other (once again, a predetermined threshold is used here), then they exert positive forces on each other. This ensures that a globally uniform sensor placement is achieved. Figure 24.4 illustrates how the VFA is used for sensor deployment.

24.3.1 Virtual Forces

We now describe the virtual forces and virtual force calculation in the VFA. In the following discussion, we use the notation introduced in the previous subsection. Let S denote the set of deployed sensor nodes, i.e. $S = \{s_1, \ldots, s_k\}$ and $|S| = k$. Let the total virtual force action on a sensor node $s_p (p = 1, \ldots, k)$ be denoted by $\vec{F}_p$. Note that $\vec{F}_p$ is a vector whose orientation is determined by the vector sum of all the forces acting on s_p. Let the force exerted on s_p by another sensor $s_q (q = 1, \ldots, k, q \neq p)$ be denoted by $\vec{F}_{pq}$. In addition to the positive and negative forces due to other sensors, a sensor s_p is also subjected to forces exerted by obstacles and areas of preferential coverage in the grid. This provides us with a convenient method to model obstacles and the need for preferential coverage.

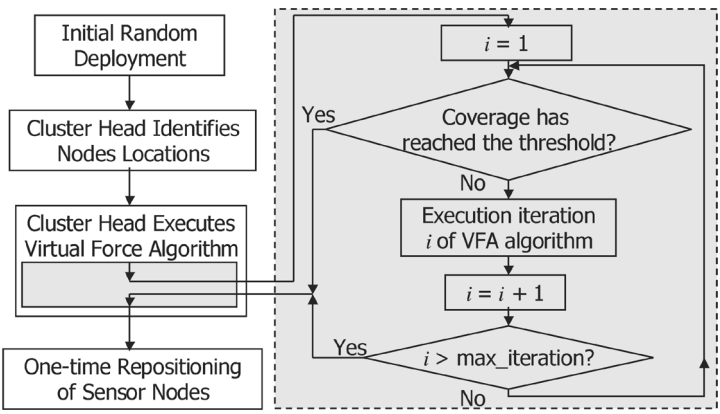


Figure 24.4. Sensor deployment with VFA.

© 2005 by Chapman & Hall/CRC

Sensor deployment must take into account the nature of the terrain, e.g. obstacles such as building and trees in the line of sight for infrared sensors, uneven surface and elevations for hilly terrain, etc. In addition, based on relative measures of security needs and tactical importance, certain areas of the grid need to be covered with greater certainty.

The knowledge of obstacles and preferential areas implies a certain degree of *a priori* knowledge of the terrain. In practice, the knowledge of obstacles and preferential areas can be used to direct the initial random deployment of sensors, which in turn can potentially increase the efficiency of the VFA. In our virtual force model, we assume that obstacles exert repulsive (negative) forces on a sensor. Likewise, areas of preferential coverage exert attractive (positive) forces on a sensor. If more detailed information about the obstacles and preferential coverage areas is available, then the parameters governing the magnitude and direction (i.e. attractive or repulsive) of these forces can be chosen appropriately. In this work, we let $\vec{F}_{p\mathrm{A}}$ be the total attractive force on s_p due to preferential coverage areas, and let $\vec{F}_{p\mathrm{R}}$ be the total repulsive force on s_p due to obstacles. The total force $\vec{F}_p$ on s_p can now be expressed as

$$\vec{F}_p = \sum_{q=1,\ q\neq p}^{k} \vec{F}_{pq} + \vec{F}_{p\mathrm{R}} + \vec{F}_{p\mathrm{A}} \tag{24.4}$$

We next express the force $\vec{F}_{pq}$ between s_p and s_q in polar coordinate notation. Note that $\vec{f} = (r, \theta)$ implies a magnitude of r and orientation θ for vector $\vec{f}$.

$$\vec{F}_{pq} = \begin{cases} (w_{\mathrm{A}}(d_{pq} - d_{\mathrm{th}}), \theta_{pq}) & \text{if } d_{pq} > d_{\mathrm{th}} \\ 0 & \text{if } d_{pq} = d_{\mathrm{th}} \\ (w_{\mathrm{R}}\dfrac{1}{d_{pq}}, \theta_{pq} + \pi) & \text{if otherwise} \end{cases} \tag{24.5}$$

where $d_{pq} = \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2}$ is the Euclidean distance between sensor s_p and s_q, d_{th} is the threshold on the distance between s_p and s_q, θ_{pq} is the orientation (angle) of a line segment from s_p to s_q, and w_{A} (w_{R}) is a measure of the attractive (repulsive) force. The threshold distance d_{th} controls how close sensors get to each other. As an example, consider the four sensors s_1, s_2, s_3 and s_4 in Figure 24.5. The force $\vec{F}_1$ on s_1 is given by $\vec{F}_1 = \vec{F}_{12} + \vec{F}_{13} + \vec{F}_{14}$. If we assume that $d_{12} > d_{\mathrm{th}}$, $d_{13} < d_{\mathrm{th}}$, and $d_{14} = d_{\mathrm{th}}$, then s_2 exerts an attractive force on s_1, s_3 exerts a repulsive force on s_1, and s_4 exerts no force on s_1. This is shown in Figure 24.5. Note that d_{th} is a predetermined parameter that is supplied by

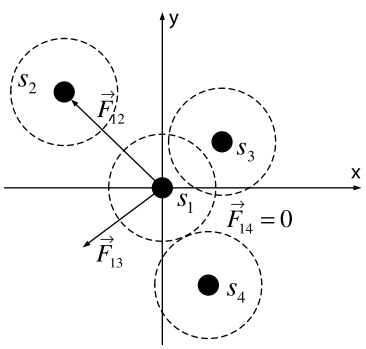


Figure 24.5. An example of virtual forces with four sensors.

© 2005 by Chapman & Hall/CRC

the user, who can choose an appropriate value of d_{th} to achieve a desired coverage level over the sensor field.

24.3.2 Overlapped Sensor Detection Areas

If $r_e \approx 0$ and we use the binary sensor detection model given by Equation (24.1), then we attempt to make d_{pq} as close to $2r$ as possible. This ensures that the detection regions of two sensors do not overlap, thereby minimizing "wasted overlap" and allowing us to cover a large grid with a small number of sensors. This is illustrated in Figure 24.6(a). An obvious drawback here is that a few grid points are not covered by any sensor. Note that an alternative strategy is to allow overlap, as shown in Figure 24.6(b). While this approach ensures that all grid points are covered, it needs more sensors for grid coverage. Therefore, we adopt the first strategy. Note that, in both cases, the coverage is effective only if the total area $k\pi r^2$ that can be covered with the k sensors exceeds the area of the grid.

If $r_e > 0$, then r_e is not negligible and the probabilistic sensor model given by Equation (24.2) or Equation (24.3) is used. Note that, owing to the uncertainty in sensor detection responses, grid points are not uniformly covered with the same probability. Some grid points will have low coverage if they are covered by only one sensor and they are far from the sensor. In this case, it is necessary to overlap sensor detection areas in order to compensate for the low detection probability of grid points that are far from a sensor. Consider a grid point with coordinate (i, j) lying in the overlap region of sensors s_p and s_q located at (x_p, y_p) and (x_q, y_q) respectively. Let $c_{ij}(s_p, s_q)$ be the probability that a target at this grid point is reported as being detected by observing the outputs of these two sensors. We assume that sensors within a cluster operate independently in their sensing activities. Thus

$$c_{ij}(s_p, s_q) = 1 - (1 - c_{ij}(s_p))(1 - c_{ij}(s_q)) \tag{24.6}$$

where $c_{ij}(s_p) = c_{ij}(x_p, y_p)$ and $c_{ij}(s_q) = c_{ij}(x_q, y_q)$ are coverage probabilities from the probabilistic sensor detection models, as we defined in Section 24.2. Since the term $1 - (1 - c_{ij}(s_p))(1 - c_{ij}(s_q))$ expresses the probability that neither s_p nor s_q covers the grid point at (i, j), the probability that the grid point (i, j) is covered is given by Equation (24.6). Let c_{th} be the desired coverage threshold for all grid points. This implies that

$$\min_{i,j}\{c_{ij}(s_p, s_q)\} \geq c_{\text{th}} \tag{24.7}$$

Note that Equation (24.6) can also be extended to a region which is overlapped by a set of k_{ov} sensors, denoted as S_{ov}, $k_{\text{ov}} = |S_{\text{ov}}|$, $S_{\text{ov}} \subseteq s_1, s_2, \ldots, s_k\}$. The coverage of the grid point at (i, j) due to a set of sensor nodes S_{ov} in this case is given by:

$$c_{ij}(S_{\text{ov}}) = 1 - \prod_{s_p \in S_{\text{ov}}} (1 - c_{ij}(s_p)) \tag{24.8}$$

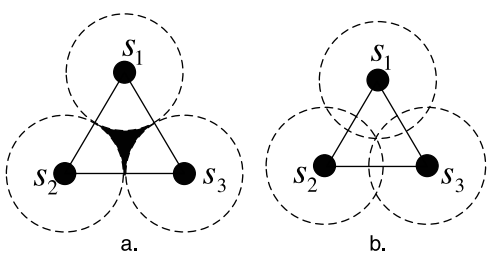


Figure 24.6. Nonoverlapped and overlapped sensor coverage areas.

© 2005 by Chapman & Hall/CRC

As shown in Equation (24.5), the threshold distance d_{th} is used to control how close sensors get to each other. When sensor detection areas overlap, the closer the sensors are to each other, the higher is the coverage probability for grid points in the overlapped areas. Note, however, that there is no increase in the point coverage once one of the sensors gets close enough to provide detection with a probability of one. Therefore, we need to determine d_{th} that maximizes the number of grid points in the overlapped area that satisfies $c_{ij}(s_p) > c_{\text{th}}$.

24.3.3 Energy Constraint on the VFA Algorithm

In order to prolong the battery life, the distances between the initial and final positions of the sensors are limited in the repositioning phase to conserve energy. We use $d_{\max}(s_p)$ to denote the maximum distance that sensor s_p can move in the repositioning phase. To simplify the discussion without loss of generality, we assume $d_{\max}(s_p) = d_{\max}(s_q) = d_{\max}$, for $p, q = 1, 2, \ldots, k$. During the execution of the VFA, for each sensor node, whenever the distance from the current virtual position to the initial position reaches the distance limit $d_{\max}$, any virtual forces on this sensor are disabled. For sensor s_p, let $(x_p, y_p)_{\text{random}}$ be the initial location obtained from the random deployment and $(x_p, y_p)_{\text{virtual}}$ be the location generated by the VFA. The energy constraint can be described as

$$\vec{F}_p = \begin{cases} 0 & \text{if } d((x_p, y_p)_{\text{random}}, (x_p, y_p)_{\text{virtual}}) \geq d_{\max} \\ \vec{F}_p & \text{otherwise (i.e. the force is unchanged)} \end{cases} \tag{24.9}$$

Therefore, the virtual force $\vec{F}_p$ given by Equation (24.4) on sensor s_p is ignored whenever the move violates the energy constraint expressed by $d_{\max}$. Note that due to the energy constraint on the one-time repositioning given by Equation (24.9), it might be necessary to trade off the coverage with the energy consumed in repositioning if $d_{\max}$ is not large enough.

Note that the VFA is designed to be executed on the cluster head, which is expected to have more computational capabilities than sensor nodes. The cluster head uses the VFA to find appropriate sensor node locations based on the coverage requirements. The new locations are then sent to the sensor nodes, which perform a one-time movement to the designated positions. No movements are performed during the execution of the VFA.

24.3.4 Procedural Description of the VFA

We next describe the VFA in pseudocode. Figure 24.7 shows the data structure of the VFA and Figure 24.8 shows the implementation details in pseudocode form. For an $n \times m$ grid with a total of k sensors deployed, the computational complexity of the VFA is $O(nmk)$. Owing to the granularity of the grid and the fact that the actual coverage is evaluated by the number of grid points that have been adequately covered, the convergence of the VFA is controlled by a threshold value, denoted by Δc. Let us use $c(loops)$ to denote the current grid coverage of the number *loops* iteration in the VFA.

For the binary sensor detection model without the energy constraint, the upper-bound value, denoted as $\bar{c}$, is $k\pi r^2$; for the probabilistic sensor detection model or binary sensor detection model with the energy constraint, $c(loops)$ is checked for saturation by defining $\bar{c}$ as the average of the coverage ratios of the near five (or ten) iterations. Therefore, the VFA continues to iterate until $|c(loops) - \bar{c}| \leq \Delta c$. In our experiments, Δc is set to 0.001.

Note that there exists the possibility of certain pathological scenarios in which the VFA is rendered ineffective, e.g. if the sensors are initially placed along the circumference of a circle such that all virtual forces are balanced. The efficiency of the VFA depends on the values of the force parameters w_{A} and w_{R}. We found that the algorithm converged more rapidly for our case studies if $w_{\text{R}} \gg w_{\text{A}}$. This need not always be true, so we are examining ways to choose appropriate values for w_{R} and w_{A} based on the initial configuration.

© 2005 by Chapman & Hall/CRC

VFA Data Structures: `Grid,` $\{s_1, s_2, \ldots, s_k\}$

/* n_P is the number of preferential area blocks (attractive forces) and n_O is the number of obstacle blocks (repulsive forces). $(x, y)_{VFA}$ is the final position found by the VFA. d_{max} is the energy constraint on the sensor repositioning phase in the VFA. */

1 *Grid structure:*
2 Properties: width, height, k, c_{th}, d_{th}, c(loops), $\bar{c}$, Δc;
3 Preferential areas: $PA_i(x, y, wx, wy)$, $i = 1, 2, \ldots, n_P$;
4 Obstacles areas: $OA_i(x, y, wx, wy)$, $i = 1, 2, \ldots, n_O$;
5 Grid points, P_{ij}: $c_{ij}(\{s_1, s_2, \ldots, s_k\})$;
6 Sensor s_p structure: $(x_p, y_p)_{random}$, $(x_p, y_p)_{virtual}$, $(x, y)_{VFA}$, p, r, r_e, α, β, d_{max}.

Figure 24.7. Data structures used in the VFA.

Procedure *Virtual_Force_Algorithm* (`Grid,` $\{s_1, s_2, \ldots, s_k\}$)

1 Set loops = 0;
2 Set MaxLoops = **MAX_LOOPS**;
3 **While** (loops > MaxLoops)
4 /* coverage evaluation */
5 **For** grid point P at (i, j) in Grid, $i \in [1, \text{width}]$, $j \in [1, \text{height}]$
6 **For** $s_p \in \{s_1, s_2, \ldots, s_k\}$
7 Calculate $c_{ij}(x_p, y_p)$ from the sensor model using $(d_{ij}(x_p, y_p), c_{th}, d_{th}, \alpha, \beta)$;
8 **End**
9 **End**
10 **If** coverage requirements are met: $|c(\text{loops}) - \bar{c}| \leq \Delta c$
11 **Break** from **While** loop;
12 **End**
13 /* virtual forces among sensors */
14 **For** $s_p \in \{s_1, s_2, \ldots, s_k\}$
15 Calculate $\vec{F}_{pq}$ using $d(s_p, s_q)$, d_{th}, w_A, w_R;
16 Calculate $\vec{F}_{pA}$ using $d(s_p, PA_1, \ldots, PA_{nP})$, d_{th};
17 Calculate $\vec{F}_{pR}$ using $d(s_p, OA_1, \ldots, OA_{nO})$, d_{th};
18 $\vec{F}_p = \sum \vec{F}_{pq} + \vec{F}_{pR} + \vec{F}_{pA}$, $q = 1, \ldots, k, q \neq p$;
19 **End**
20 /*move sensors virtually */
21 **For** $s_p \in \{s_1, s_2, \ldots, s_k\}$
22 /* energy constraint on the sensor movement */
23 **If** $d((x_p, y_p)_{random}, (x,y)_{virtual}) \geq d_{max}$
24 Set $\vec{F}_p = 0$;
25 **End**
26 $\vec{F}_p$ virtually moves s_p to its next position;
27 **End**
28 /* continue to next iteration */
29 Set loops = loops + 1;
30 **End**

Figure 24.8. Pseudo code of the VFA algorithm.

© 2005 by Chapman & Hall/CRC

24.3.5 VFA Simulation Results

In this section we present simulation results obtained using the VFA. The deployment requirements include the maximum improvement of coverage over random deployment, the coverage for preferential areas and the avoidance of obstacles. For all simulation results presented in this section, distances are measured in units of grid points. A total of 20 sensors are placed in the sensor field in the random placement stage. Each sensor has a detection radius of five units ($r = 5$) and range detection error of three units ($r_e = 3$) for the probabilistic detection model. The sensor field is 50×50 in dimension. The simulation is done on a Pentium III 1.0 GHz PC using Matlab.

24.3.6 Case Study 1

Figures 24.9–24.11 present simulation results for the probabilistic sensor model given by Equation (24.3). The probabilistic sensor detection model parameters are set as $\lambda = 0.5$, $\beta = 0.5$, and $c_{th} = 0.7$. The initial sensor placements are shown in Figure 24.9. Figure 24.10 shows the final sensor positions determined by the VFA. Figure 24.11 shows the virtual movement traces of all sensors during the execution of the VFA. We can see that overlap areas are used to increase the number of grid points whose coverage exceeds the required threshold c_{th}. Figure 24.12 shows the improvement of coverage during the execution of the VFA.

24.3.7 Case Study 2

As discussed in Section 24.3, the VFA is also applicable to a sensor field containing obstacles and preferential areas. If obstacles are to be avoided, then they can be modeled as repulsive force sources in the VFA. Preferential areas should be covered first; then, therefore, they are modeled as attractive force sources in the VFA. Figures 24.13–24.16 present simulation results for a 50×50 sensor field

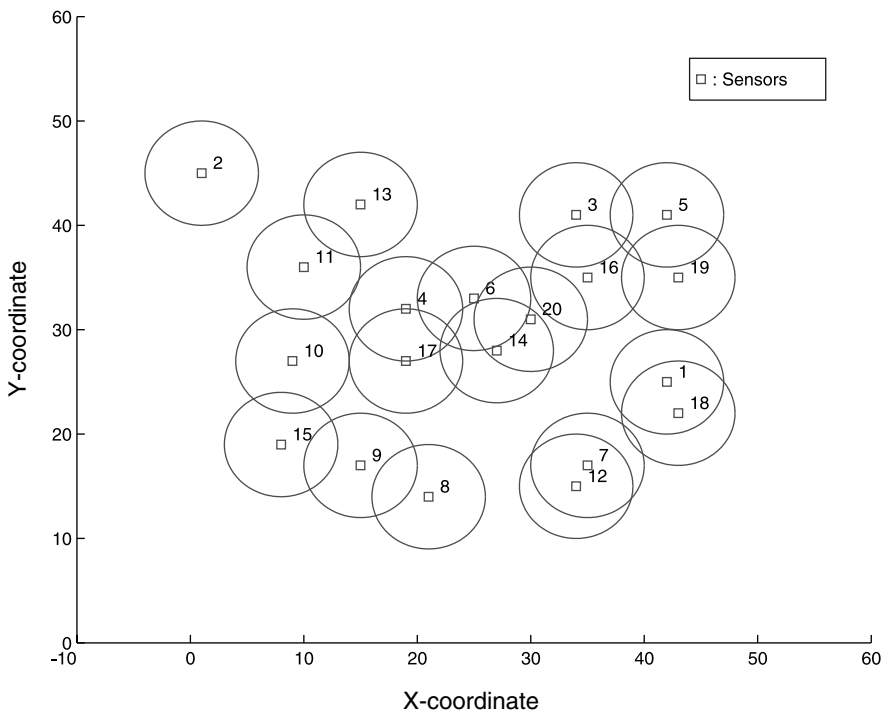


Figure 24.9. Initial sensor positions after random placement (probabilistic sensor detection model).

© 2005 by Chapman & Hall/CRC

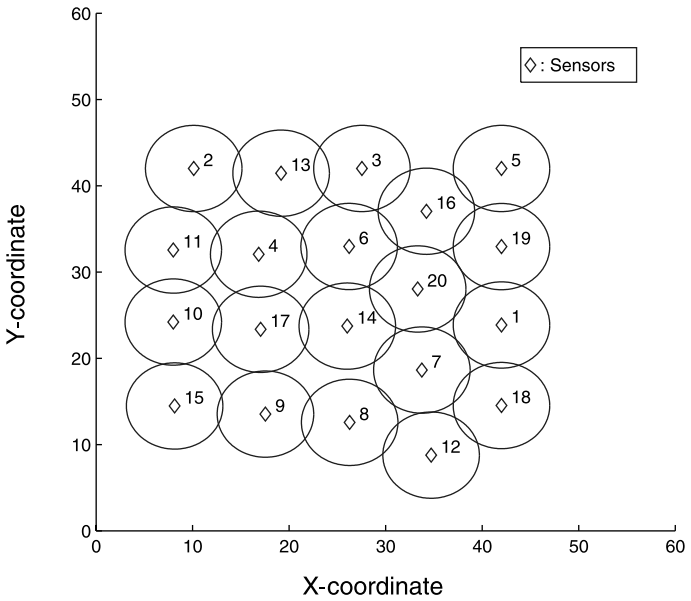


Figure 24.10. Sensor positions after the execution of the VFA (probabilistic sensor detection model).

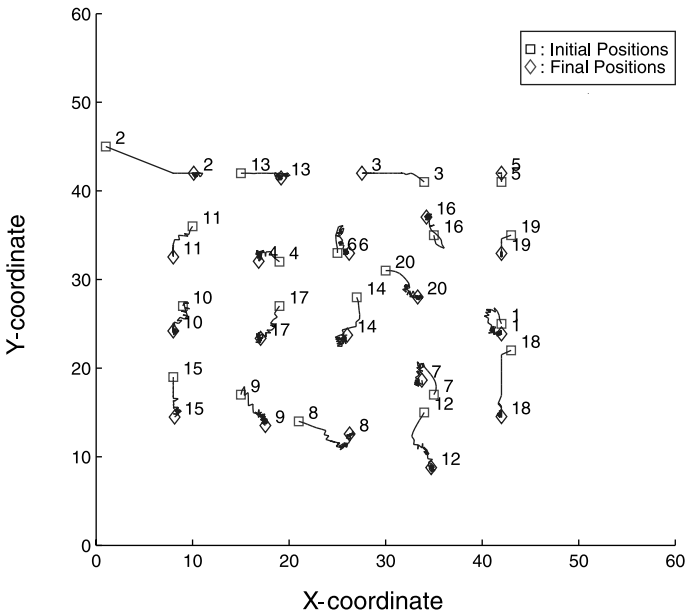


Figure 24.11. A trace of virtual moves made by the sensors (probabilistic sensor detection model).

that contains an obstacle and a preferential area. The binary sensor detection model given by Equation (24.1) is used for this simulation. The initial sensor placements are shown in Figure 24.13. Figure 24.14 shows the final sensor positions determined by the VFA. Figure 24.15 shows the virtual movement traces of all sensors during the execution of the VFA. Figure 24.16 shows the improvement of coverage during the execution of the VFA. The VFA does not require much computation time. For case study 1, the VFA took only 25 s for 30 iterations. For case study 1, the VFA took only 3 min to complete 50 iterations. Finally, for case study 2, the VFA took only 48 s to complete 50 iterations.

© 2005 by Chapman & Hall/CRC

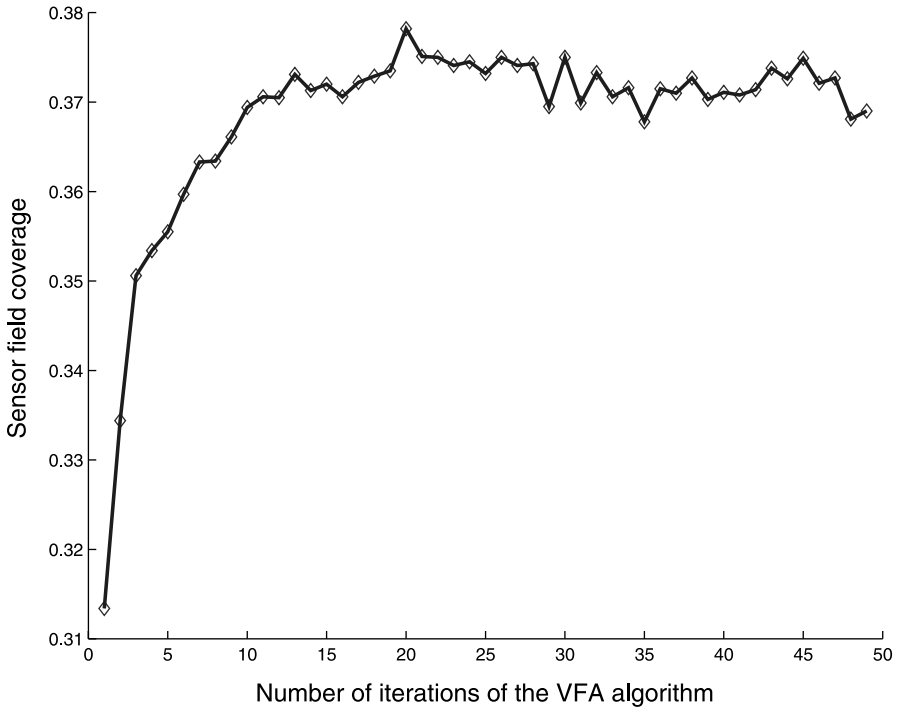


Figure 24.12. Sensor field coverage achieved using the VFA (probabilistic sensor detection model).

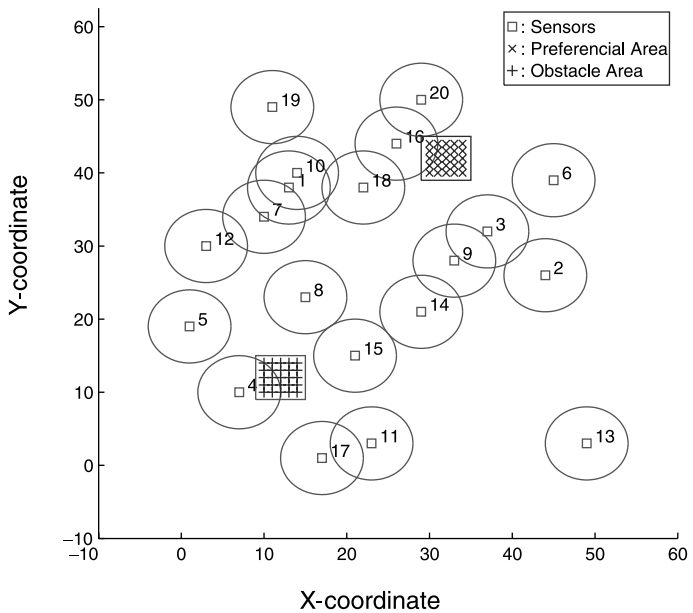


Figure 24.13. Initial sensor positions after random placement with obstacles and preferred areas.

Note that these computation times include the time needed for displaying the simulation results on the screen. CPU time is important because sensor redeployment should not take excessive time.

In order to examine how the VFA scales for larger problem instances, we considered up to 90 sensor nodes in a cluster for a 50×50 grid, with $r = 3$, $r_e = 2$, $\lambda = 0.5$ and $\beta = 0.5$ for all cases. For a given

© 2005 by Chapman & Hall/CRC

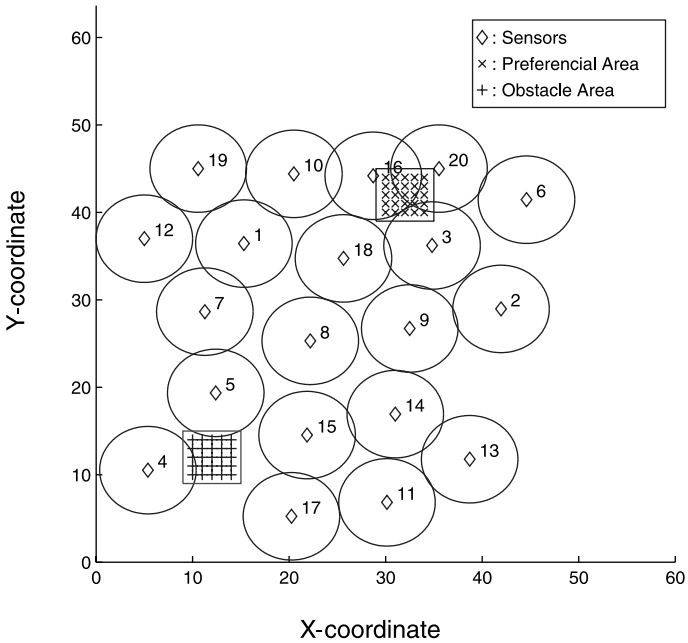


Figure 24.14. Sensor positions after the execution of the VFA with obstacles and preferred areas.

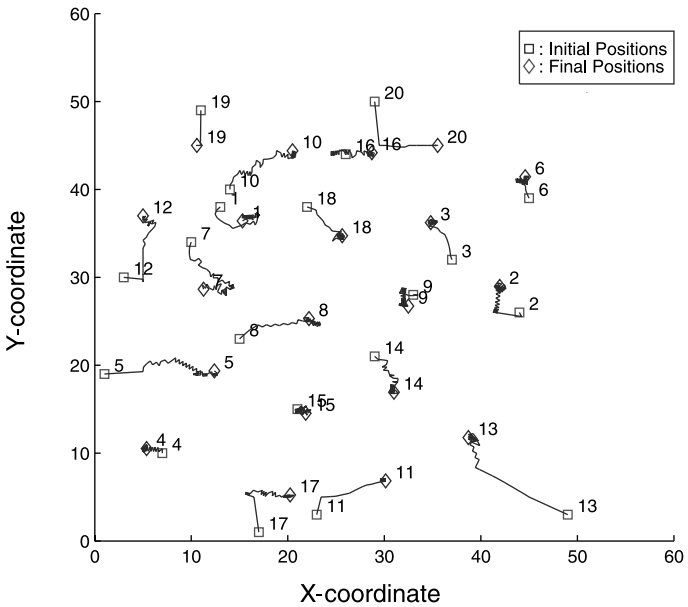


Figure 24.15. A trace of virtual moves made by the sensors with obstacles and preferred areas.

number of sensor nodes, we run the VFA over ten sets of random deployment results and take the average of the computation time. The results, listed in Table 24.1, show that the CPU time grows slowly with the number of sensors k. For a total of 90 sensors, the CPU time is only 4 min on a Pentium III PC. In practice, a cluster head usually has less computational power than a Pentium III PC; however, our results indicate that even if the cluster head has less memory and an on-board processor that runs ten times slower, the CPU time for the VFA is reasonable.

© 2005 by Chapman & Hall/CRC

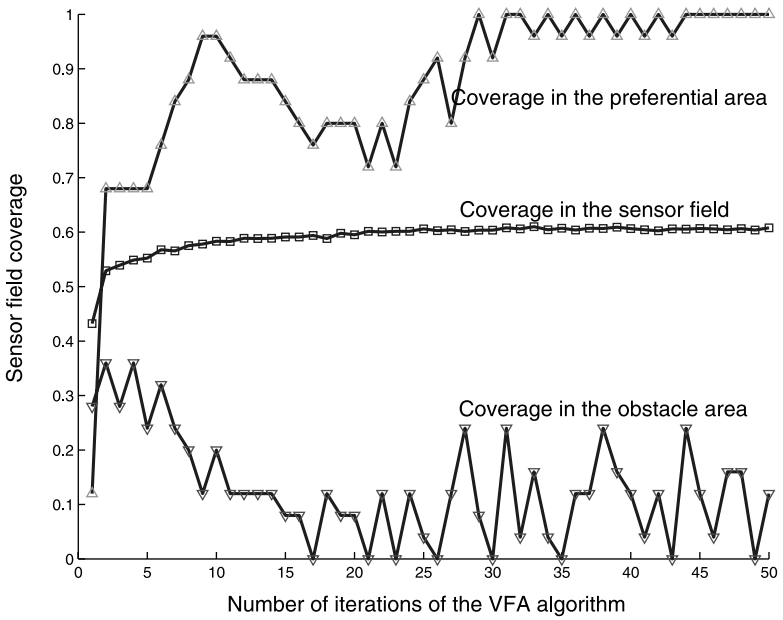


Figure 24.16. Sensor field coverage achieved using the VFA with obstacles and preferred areas.

Table 24.1. The computation time for the VFA for larger problem instances

k	Binary model	Probabilistic model	k	Binary model	Probabilistic model
40	21 s	1.8 min	70	46 s	3.6 min
50	32 s	2.2 min	80	59 s	3.7 min
60	38 s	3.1 min	90	64 s	4.0 min

24.4 Uncertainty Modeling in Sensor Node Deployment

The topology of the sensor field, i.e. the locations of the sensors, determines to a large extent the quality and the extent of the coverage provided by the sensor network. However, even if the sensor locations are precomputed for optimal coverage and resource utilization, there are inherent uncertainties in the sensor locations when the sensors are dispersed, scattered, or airdropped. Thus, a key challenge in sensor deployment is to determine an uncertainty-aware sensor field architecture that reduces cost and provides high coverage, even though the exact location of the sensors may not be controllable. We consider the sensor deployment problem in the context of uncertainty in sensor locations subsequent to airdropping. Sensor deployment in such scenarios is inherently nondeterministic, and there is a certain degree of randomness associated with the location of a sensor in the sensor field. We present two algorithms for the efficient placement of sensors in a sensor field when the exact locations of the sensors are not known.

In applications such as battlefield surveillance and environmental monitoring, sensors may be dropped from airplanes. Such sensors cannot be expected to fall exactly at predetermined locations; rather, there are regions where there is a high probability of a sensor being actually located (Figure 24.17). In underwater deployment, sensors may move due to drift or water currents. Furthermore, in most real-life situations, it is difficult to pinpoint the exact location of each sensor since only a few of the sensors may be aware of their locations. Thus, the positions of sensors may not be known exactly, and for every point in the sensor field there is only a certain probability of a sensor being located at that point.

© 2005 by Chapman & Hall/CRC

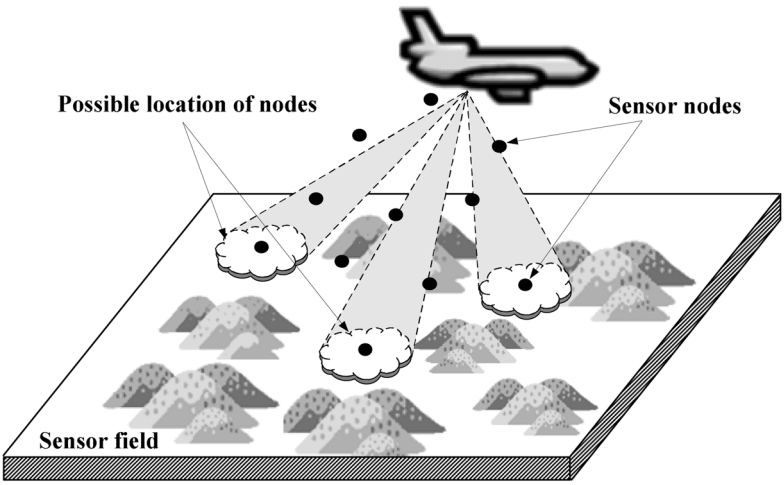


Figure 24.17. Sensors dropped from airplanes. The clouded region gives the possible region of a sensor location. The black dots within the clouds show the mean (intended) position of a sensor.

In this section, we present two algorithms for sensor deployment wherein we assumed that sensor positions are not exactly predetermined. We assume that the sensor locations are calculated before deployment and an attempt is made during the airdrop to place sensors at these locations; however, the sensor placement calculations and coverage optimization are based on a Gaussian model, which assumes that if a sensor is intended for a specific point P in the sensor field, then its exact location can be anywhere in a ''cloud'' surrounding P.

24.4.1 Modeling of Nondeterministic Sensor Node Placement

During sensor deployment, an attempt is made to place sensors at appropriate predetermined locations by airdropping or other means. This does not guarantee, however, that sensors are actually placed at the designated positions, due to unanticipated conditions such as wind, the slope of the terrain, etc. In this case, there is a certain probability of a sensor being located at a particular grid point as a function of the designated location. The deviation about the designated sensor locations may be modeled using a Gaussian probability distribution, where the intended coordinates (x, y) serve as the mean values with standard deviation σ_x and σ_y in the x and y dimensions respectively. Assuming that the deviations in the x and y dimensions are independent, the joint probability density function with mean (x, y) is given by

$$p_{xy}(x', y') = \frac{\exp\left[(-(x' - x)^2/2\sigma_x^2) - ((y' - y)^2/2\sigma_y^2)\right]}{2\pi\sigma_x\sigma_y} \tag{24.10}$$

Let us use the notation introduced in the previous section. We still consider a sensor field represented by an $m \times n$ grid, denoted as *Grid*, with S denoting the set of sensor nodes. Let L_S be the set that contains corresponding sensor node locations, i.e. $L_S = \{(x_p, y_p) | s_p \text{ at } (x_p, y_p), s_p \in S\}$. Let A be the total area encompassing all possible sensor locations. To model the uncertainty in sensor locations, the conditional probability $c_{ij}^*(x, y)$ for a grid point (i, j) to be detected by a sensor that is supposed to be deployed at (x, y) is then given by

$$c_{ij}^*(x, y) = \frac{\sum_{(x', y') \in A} c_{ij}(x', y') p_{xy}(x', y')}{\sum_{(x', y') \in A} p_{xy}(x', y')} \tag{24.11}$$

© 2005 by Chapman & Hall/CRC

Based on Equations (24.10) and (24.11), we define the matrices $C^*_{xy} = [c^*_{ij}(x,y)]_{m\times n}$ and $P = [p_{xy}(x',y')]_A$.

24.4.2 Uncertainty-Aware Sensor Node Placement Algorithms

In this section we introduce the sensor placement algorithm with consideration of uncertainties in sensor locations. The goal of sensor placement algorithms is to determine the minimum number of sensors and their locations such that every grid point is covered with a minimum confidence level. The sensor placement algorithms do not give us the actual location of the sensor, only the mean position of the sensor. It is straightforward to define the *miss probability* in our sensor deployment scenario. The miss probability of a grid point (i,j) due to a sensor at (x,y), denoted as $m_{ij}(x,y)$, is given by

$$m_{ij}(x,y) = 1 - c^*_{ij}(x,y) \tag{24.12}$$

Therefore, the miss probability matrix due to a sensor placed at (x,y) is $M_{xy} = [m_{ij}(x,y)]_{m\times n}$. M_{xy} is associated with each grid point and can be predetermined based on Equations (24.10)–(24.12). Since a number of sensors are placed for coverage, we would like to know the miss probability of each grid point due to a set of sensors, namely the *collective miss probability*. We denote the term collective miss probability as m_{ij} and define it in the form of a maximum likelihood function as

$$m_{ij} = \prod_{(x,y)\in L_S} m_{ij}(x,y) = \prod_{(x,y)\in L_S} [1 - c^*_{ij}(x,y)] \tag{24.13}$$

Accordingly we have $M = [m_{ij}]_{m\times n}$ as the collective miss probability matrix over the grid points in the sensor field.

We determine the location of the sensors one at a time. In each step, we find all possible locations that are available on the grid for a sensor, and calculate the *overall miss probability* associated due to this sensor and those already deployed. We denote the overall miss probability due to the newly introduced sensor at grid point (x,y) as $\widetilde{m}(x,y)$, which is defined as

$$\widetilde{m}(x,y) = \sum_{(i,j)\in Grid} m_{ij}(x,y) m_{ij} \tag{24.14}$$

Based on the $\widetilde{m}(x,y)$ values, where $(x,y) \in Grid$ and $(x,y) \notin L_S$, we can place sensors either at the grid point with the maximum miss probability (the worst coverage case) or the minimum miss probability (the best coverage case). We refer to the two strategies as MAX_MISS and MIN_MISS respectively. Therefore, the sensor location can be found based on the following rule. For $(x,y) \in Grid$ and $(x,y) \notin L_S$:

$$\{\widetilde{m}(x,y) = \begin{cases} \min\{\widetilde{m}(x',y')\} & \text{if MIN_MISS is used} \\ \max\{\widetilde{m}(x',y')\} & \text{if MAX_MISS is used} \end{cases} \tag{24.15}$$

When the best location is found for the current sensor, the collective miss probability matrix M is updated with the newly introduced sensor at location (x,y). This is carried out using Equation (24.16):

$$M = M\, M_{xy} = [m_{ij}\, m_{ij}(x,y)]_{m\times n} \tag{24.16}$$

There are two parameters that serve as the termination criterion for the two algorithm. The first is $k_{\max}$, which is the maximum number of sensors that we can afford to deploy. The second is the threshold on the miss probability of each grid point, m_{th}. Our objective is to ensure that every grid

© 2005 by Chapman & Hall/CRC

point is covered with probability at least $c_{\text{th}} = 1 - m_{\text{th}}$. Therefore, the rule to stop the further execution of the algorithm is

$$m_{ij} < m_{\text{th}} \quad \text{for all } (i,j) \in \mathit{Grid} \textbf{ or } k > k_{\max} \tag{24.17}$$

where k is the number of sensors deployed. The performance of the proposed algorithm is evaluated using the *average coverage probability* of the grid, defined as

$$c_{\text{avg}} = \frac{\sum_{(x,y)\in \mathit{Grid}} c_{ij}}{m\,n} \tag{24.18}$$

where c_{ij} is the *collective coverage probability* of a grid point due to all sensors on the grid, defined as

$$\begin{aligned} c_{ij} &= 1 - \prod_{(x,y)\in L_S} m_{ij}(x,y) \\ &= 1 - \left\{ \prod_{(x,y)\in L_S} [1 - c_{ij}^{*}(x,y)] \right\} \end{aligned} \tag{24.19}$$

We have thus far only considered the coverage of the grid points in the sensor field. In order to provide robust coverage of the sensor field, we also need to ensure that the region that lies between the grid points is adequately covered, i.e. every nongrid point has a miss probability less than the threshold m_{th}. Consider the four grid points in Figure 24.18 that lie on the four corners of a square. Let the distance between these grid points be d^{*}. The point of intersection of the diagonals of the square is at distance $d^{*}/\sqrt{2}$ from the four grid points. The following theorem provides a sufficient condition under which the nongrid points are adequately covered by the MIN_MISS and MAX_MISS algorithms:

Theorem 24.1. Let the distance between the grid point P_1 and a potential sensor location P_2 be d. Let the distance between adjacent grid points be d^{*}. If a value of $d + (d^{*}/\sqrt{2})$ is used to calculate the coverage of grid point P_1 due to a sensor at P_2, and the number of available sensors is adequate, then the miss probability of all the nongrid points is less than a given threshold m_{th} when the algorithms MAX_MISS and MIN_MISS terminate.

Proof. Consider the four grid points in Figure 24.18. The center of the square, i.e. the point of intersection of diagonals, is at a distance of $d^{*}/\sqrt{2}$ from each of the four grid points. Every other

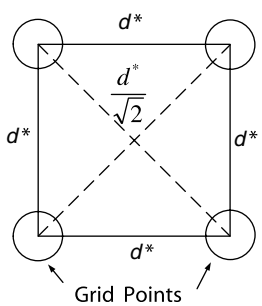


Figure 24.18. Coverage of nongrid points.

© 2005 by Chapman & Hall/CRC

nongrid point is at a shorter distance (less than $d^*/\sqrt{2}$) from at least one of the four grid points. Thus, if a value of $d + (d^*/\sqrt{2})$ is used to determine coverage in the MAX_MISS and MIN_MISS algorithms, we can guarantee that every nongrid point is covered with a probability that exceeds $1 - m_{\text{th}}$. □

In order to illustrate Theorem 24.1, we consider a 5×5 grid with $\alpha = 0.5, \lambda = 0.5, \beta = 0.5$ and $m_{\text{th}} = 0.4$. We use Theorem 24.1 and the MAX_MISS algorithm to determine sensor placement and to calculate the miss probabilities for all the centers of the squares. The results are shown in Figure 24.19 and Figure 24.20 for both sensor detection models. They indicate that the miss probabilities are always less than the threshold m_{th}, thereby ensuring adequate coverage of the nongrid points.

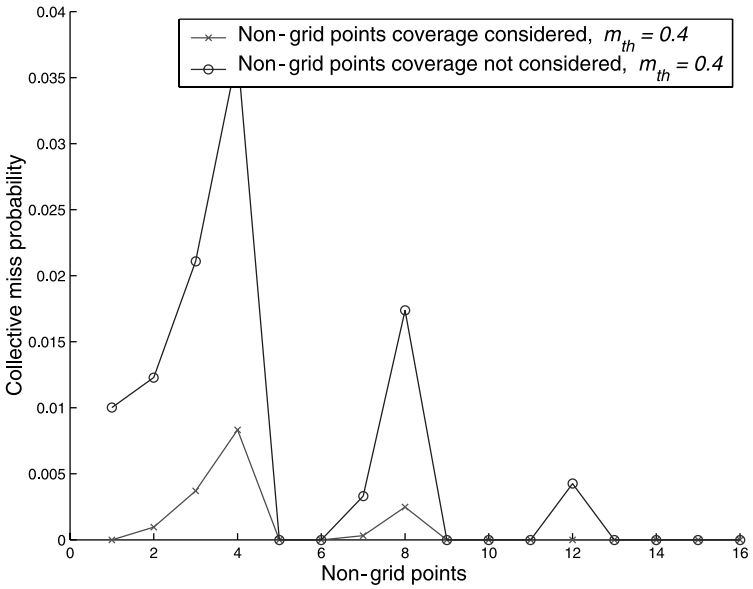


Figure 24.19. Coverage of nongrid points for the sensor model given by Equation (24.2).

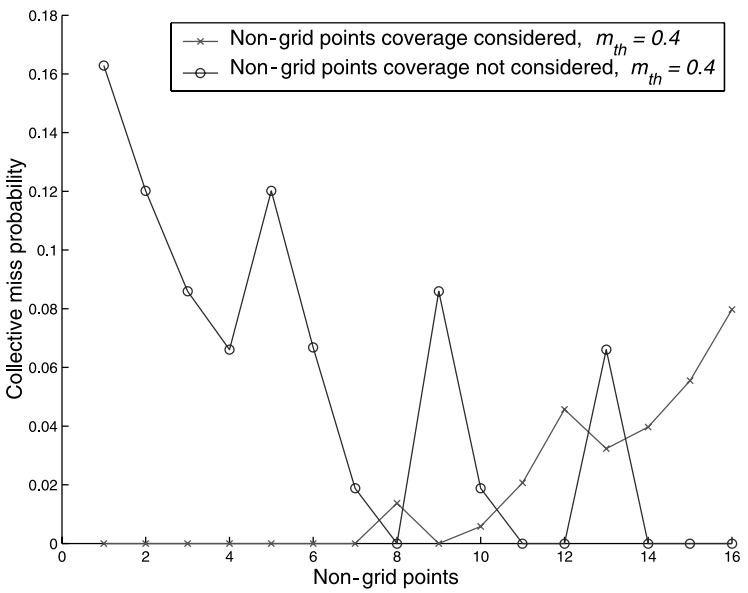


Figure 24.20. Coverage of nongrid points for the sensor model given by Equation (24.3).

© 2005 by Chapman & Hall/CRC

24.4.3 Procedural description

Note that matrices C_{xy}, M_{xy} and P_A can all be calculated before the actual execution of the placement algorithms. This is illustrated in Figure 24.21 as the pseudocode for the initialization procedure. The initialization procedure is the algorithm overhead, which has a complexity of $O((mn)^2)$, where the dimension of the grid is $m \times n$. Once the initialization is done, we may apply either the MIN_MISS or MAX_MISS uncertainty-aware sensor placement algorithm using different values for m_{th} and $k_{\max}$ with the same C_{xy}, M_{xy} and P_A. Figure 24.22 outlines the main part in pseudocode for the

Procedure *NDSP_Proc_Init* (`Grid`, $\sigma_x, \sigma_y, \alpha, \lambda, \beta$)

01 /* Build the uncertainty area matrix $P = [p_{xy}(x', y')]_A$ */
02 **For** $(x', y') \in A$
03 $\quad p_{xy}(x', y') = \dfrac{e^{-((x'-x)^2/2\sigma_x^2)-((y'-y)^2/2\sigma_y^2)}}{2\pi\sigma_x\sigma_y}$
04 **End**
05 /* Build the miss probability matrix for all grid points. */
06 **For** grid point $(x, y) \in Grid$
07 $\quad$ /* Build C_{xy}, C^*_{xy}, and M_{xy} for sensor node at (x, y). */
08 $\quad$ **For** grid point $(i, j) \in Grid$
09 $\quad\quad$ /* Non-grid points coverage are considered based on Theorem 1. */
09 $\quad\quad$ $d_{ij}(x, y) = \sqrt{(x-i)^2 + (y-j)^2} + \frac{d^*}{\sqrt{2}}$;
10 $\quad\quad$ /* Calculate the grid point coverage probability based on the sensor detection model. */
11 $\quad\quad$ Calculate $c_{ij}(x, y)$:
12 $\quad\quad$ /* Sensor detection model 1. */
13 $\quad\quad$ Model 1: $c_{ij}(x, y) = e^{-\alpha d_{ij}(x,y)}$
14 $\quad\quad$ /* Sensor detection model 2. */
15 $\quad\quad$ Model 2: $c_{ij}(x, y) = \begin{cases} 0, & if \quad r + r_e \le d_{ij}(x, y) \\ e^{-\lambda\alpha^\beta}, & if \quad |r - d_{ij}(x, y)| < r_e \\ 1, & if \quad r - r_e \ge d_{ij}(x, y) \end{cases}$
16 $\quad\quad$ /* Modeling of uncertainty in sensor node locations. */
17 $\quad\quad$ $c^*_{ij}(x, y) = \dfrac{\sum_{(x',y')\in A} c_{ij}(x', y') p_{xy}(x', y')}{\sum_{(x',y')\in A} p_{xy}(x', y')}$;
18 $\quad\quad$ /* The miss probability matrix */
19 $\quad\quad$ $m_{ij}(x, y) = 1 - c^*_{ij}(x, y)$;
20 $\quad$ **End**
21 $\quad$ /* Use the obstacle mask matrix based on the a priori knowledge about the terrain. */
22 $\quad$ **If** Obstacles exist
23 $\quad\quad$ $C_{xy} = C_{xy} \cdot$ *Obstacle Mask Matrix*
24 $\quad\quad$ Revise M_{xy}.
25 $\quad$ **End**
26 **End**
27 /* Initially the overall miss probability matrix is set to I. */
28 $M = [m_{ij}]_{m\times n} = [1]_{m\times n}$;

Figure 24.21. Initialization pseudocode.

© 2005 by Chapman & Hall/CRC

Procedure *NDSP_Proc_Main* (type, k_{max}, m_{th}, Grid, C_{xy}, M_{xy}, P_A, M)

01 /* Initially no sensors have been placed yet. */
02 Set $S = \phi$; $L_S = \{\phi\}$; $k = |S|$;
03 /* Repeatedly placing sensors until requirement is satisfied.*/
04 **Repeat**
05 /* Evaluate the miss probability due to a sensor at (x, y). */
06 **For** grid point $(x, y) \in Grid$ **and** $(x, y) \notin L_S$
07 Retrieve $M_{xy} = [m_{ij}(x, y)]_{m \times n}$
$= [1 - c^*_{ij}(x, y)]_{m \times n}$
$$= \left[1 - \frac{\sum_{(x',y') \in A} c_{ij}(x', y') p_{xy}(x', y')}{\sum_{(x',y') \in A} p_{xy}(x', y')} \right]_{m \times n}$$
08 /* Miss probability if sensor node is placed at (x, y) */
09 $\tilde{m}(x, y) = \sum_{(i,j) \in Grid} m_{ij}(x, y) m_{ij}$;
10 **End**
11 /* Place sensor node using selected algorithm. */
12 **If** *type* = MIN_MISS
13 Find $(x, y) \in Grid$ and $(x, y) \notin L_s$ such that $\tilde{m}(x, y) = \min\{\tilde{m}(x', y')\}$, $(x', y') \in Grid$;
14 **Else** /* MAX_MISS */
15 Find $(x, y) \in Grid$ and $(x, y) \notin L_s$ such that $\tilde{m}(x, y) = \max\{\tilde{m}(x', y')\}$, $(x', y') \in Grid$;
16 **End**
17 /* Save the information of sensor node just placed. */
18 Set $k = k + 1$;
19 Set $L_S = L_S \cup \{(x, y)\}$;
20 Set $S = S \cup \{s_k\}$;
20 /* Update current overall miss probability matrix. */
21 **For** grid point $(i, j) \in Grid$
22 $m_{ij} = m_{ij} \cdot m_{ij}(x, y)$;
23 **End**
24 /* Check if the placement requirement is satisfied. */
25 **Until** $m_{ij} < m_{th}$ for all $(i, j) \in Grid$ **Or** $k > k_{max}$;

Figure 24.22. Pseudocode for sensor placement algorithm.

uncertainty-aware sensor placement algorithms. The computational complexity for both MIN_MISS and MAX_MISS is $O(mn)$.

24.4.4 Simulation Results on Uncertainty-Aware Sensor Deployment

Next, we present simulation results for the proposed uncertainty-aware sensor placement algorithms MIN_MISS and MAX_MISS using the same testing platform. Note that, for practical reasons, we use a truncated Gaussian model because the sensor deviations in a sensor location are unlikely to span the complete sensor field. Therefore, $x' - x$ and $y' - y$ in Equation (24.10) are limited to a certain range, which reflects how large the variation is in the sensor locations during the deployment. The maximum error in the x direction is denoted as $e^x_{\max} = \max(x' - x)$, and the maximum error in y direction is denoted as $e^y_{\max} = \max(y' - y)$. We then present our simulation results

© 2005 by Chapman & Hall/CRC

for different sets of parameters in grid of point units, where $m = n = 10$, $\sigma_x = \sigma_y = 0.1, 0.32, 1, 2$, and $e^x_{\max} = e^y_{\max} = 2, 3, 5$.

24.4.4.1 Case Study 1

We first consider the probabilistic sensor detection model given by Equation (24.2) with $\alpha = 0.5$. Figure 24.23 presents the result for the two sensor placement algorithms described by Equation (24.15). Figure 24.24 compares the proposed MIN_MISS and MAX_MISS algorithms with the base case where no location errors are considered, i.e. an uncertainty-oblivious (UO) strategy is followed by setting $\sigma_x = \sigma_y = 0$. We also consider a random deployment of sensors. The results show that MIN_MISS is nearly as efficient as the base UO algorithm, yet it is much more robust. Figure 24.25 presents results for the truncated Gaussian models with different maximum errors. Compared with random deployment, MIN_MISS requires more sensors here, but we expect random deployment to perform worse in the presence of obstacles. Figure 24.26 is for MIN_MISS and MAX_MISS with coverage obtained without location uncertainty. The results show that the MAX_MISS algorithm, which places more sensors for a given coverage threshold, provides higher overall coverage.

24.4.4.2 Case Study 2

Next, we consider the probabilistic sensor detection model given by Equation (24.3) with $r = 5$, $r_e = 4$, $\lambda = 0.5$, and $\beta = 0.5$. Figure 24.27 presents the result for the two sensor placement algorithms described by Equation (24.15). Figure 24.28 compares the proposed MIN_MISS and MAX_MISS algorithms with the base case where no location errors are considered. Figure 24.29 presents results for the truncated Gaussian models with different maximum errors. Figure 24.30

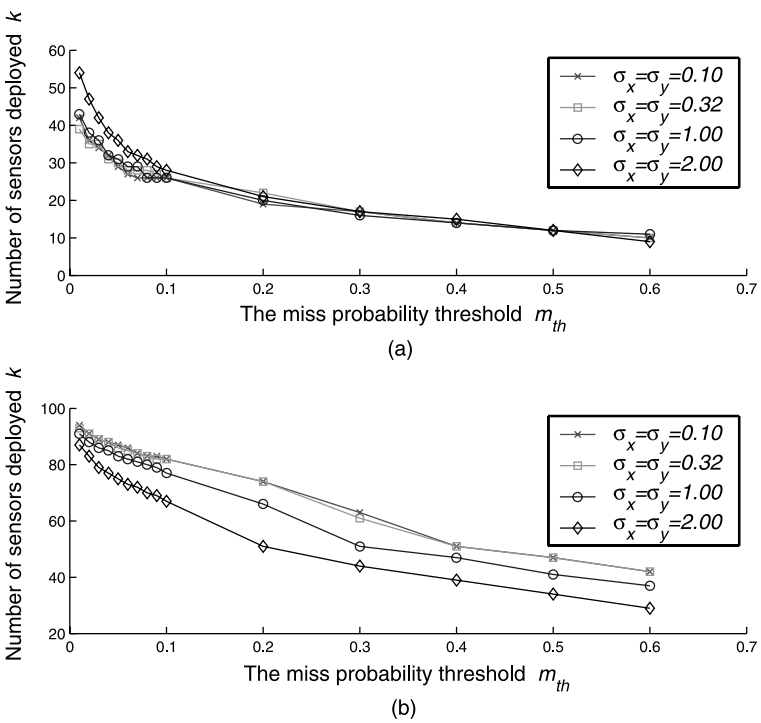


Figure 24.23. Number of sensors required as a function of the miss probability threshold with $\alpha = 0.5, e^x_{\max} = e^y_{\max} = 5$ for (a) MIN_MISS and (b) MAX_MISS.

© 2005 by Chapman & Hall/CRC

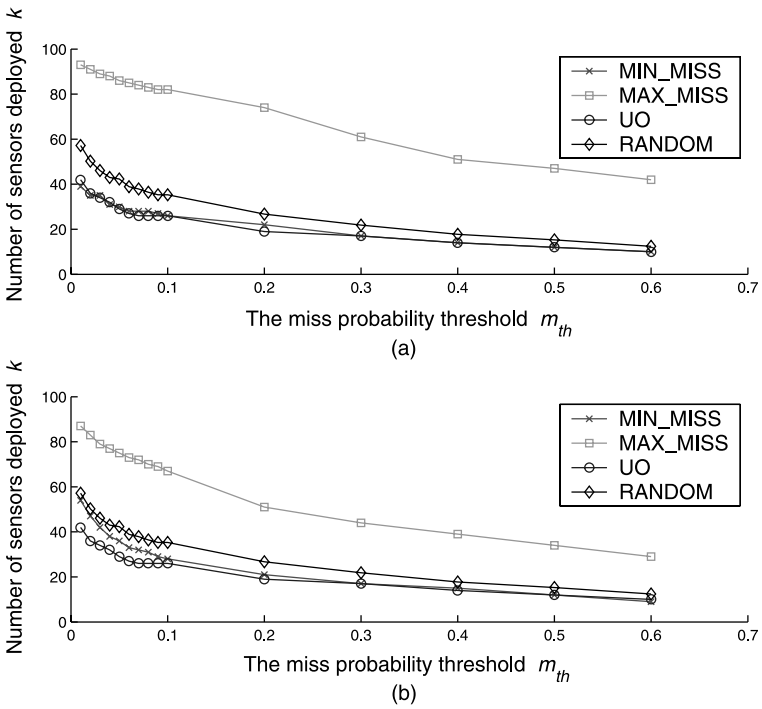


Figure 24.24. Number of sensors required for various placement schemes with $\alpha = 0.5, e^x_{\max} = e^y_{\max} = 5$, with (a) $\sigma_x = \sigma_y = 0.32$ and (b) $\sigma_x = \sigma_y = 2$.

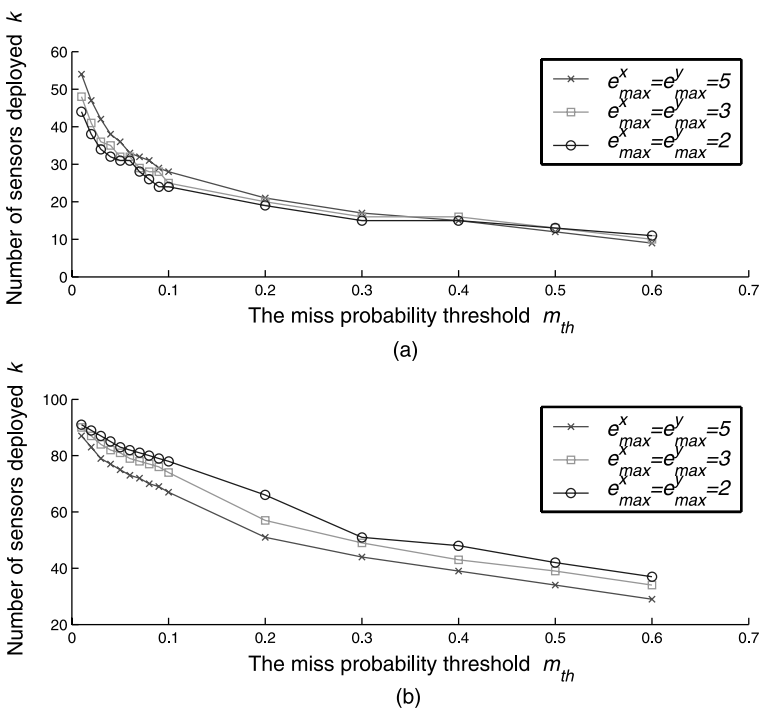


Figure 24.25. Comparisons of different truncated Gaussian models with $\alpha = 0.5, \sigma_x = \sigma_y = 2$ for (a) MIN_MISS and (b) MAX_MISS.

© 2005 by Chapman & Hall/CRC

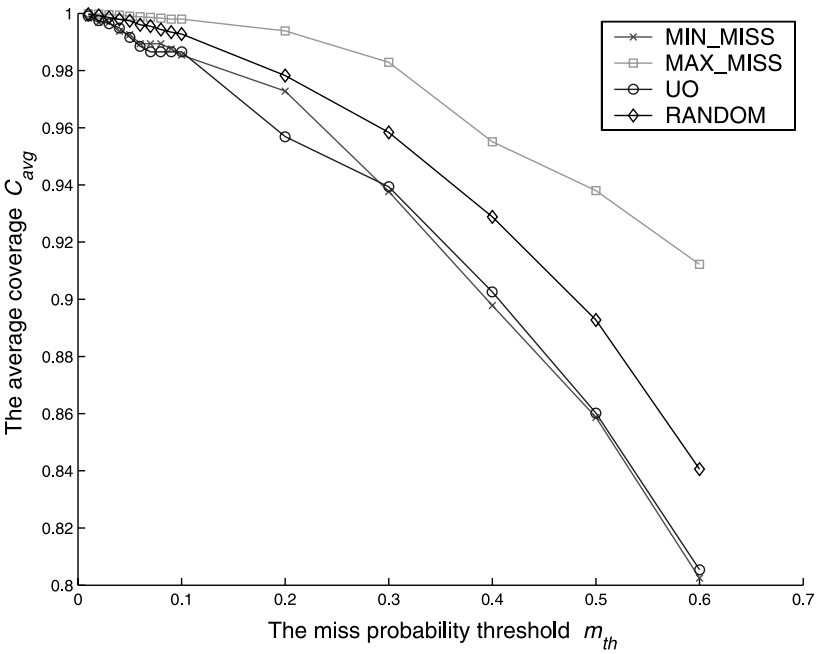


Figure 24.26. Comparison of average coverage for various placement schemes with $\alpha = 0.5, e^x_{\max} = e^y_{\max} = 5$, $\sigma_x = \sigma_y = 0.32$.

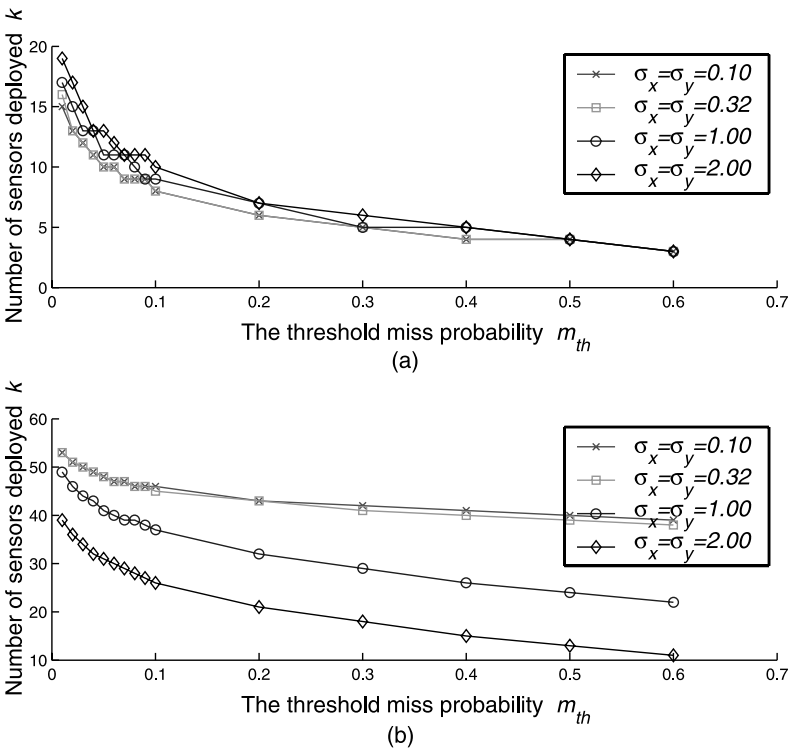


Figure 24.27. Number of sensors required as a function of the miss probability threshold with $\alpha = 0.5, e^x_{\max} = e^y_{\max} = 5$ for (a) MIN_MISS and (b) MAX_MISS.

© 2005 by Chapman & Hall/CRC

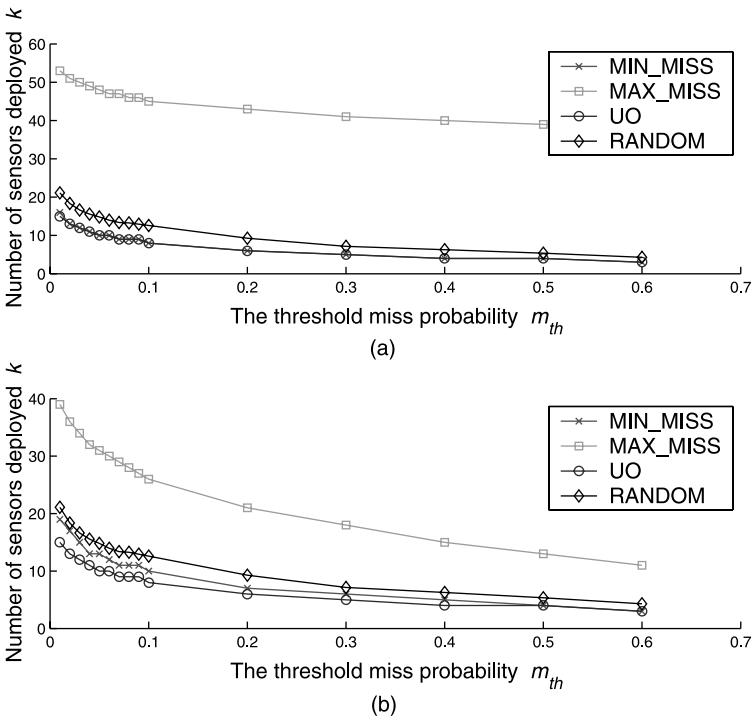


Figure 24.28. Number of sensors required for various placement schemes with $\alpha = 0.5, e^x_{\max} = e^y_{\max} = 5$, with (a) $\sigma_x = \sigma_y = 0.32$ and (b) $\sigma_x = \sigma_y = 2$.

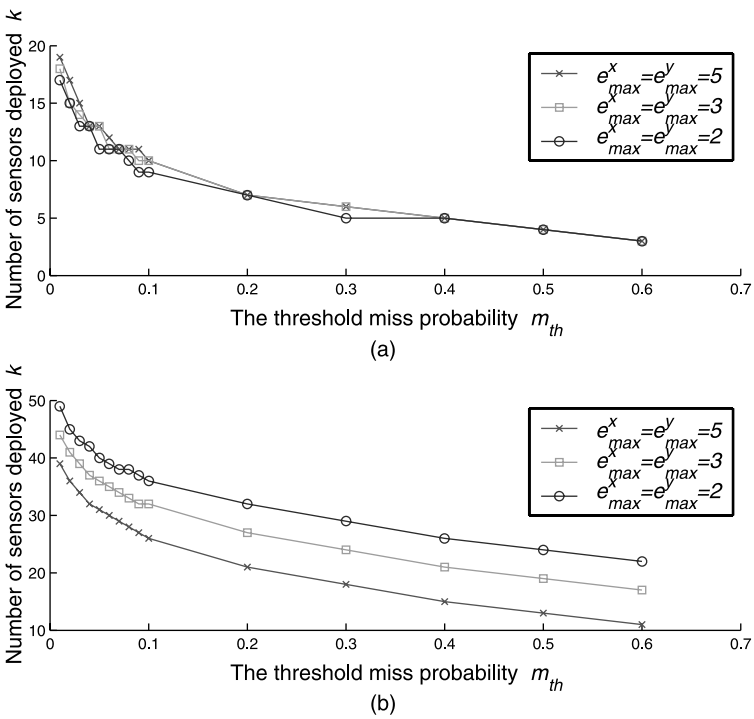


Figure 24.29. Comparisons of different truncated Gaussian models with $\alpha = 0.5, \sigma_x = \sigma_y = 2$ for (a) MIN_MISS and (b) MAX_MISS.

© 2005 by Chapman & Hall/CRC

compares the coverage based on Equation (24.18) for MIN_MISS and MAX_MISS with coverage obtained without location uncertainty. We notice that, owing to the different probability values as a reflection of the confidence level in sensor responses from these two different models, the results in sensor placement are also different. Compared with case study 1, this sensor detection model with the selected model parameters as $\lambda = 0.5$ and $\beta = 0.5$ requires a lesser number of sensor nodes for the same miss probability threshold. Part of the reason is due to the fact that, in Equation (24.3), we have full confidence in sensor responses for grid points that are very close to the sensor node, i.e. $c_{ij}(x, y) = 1$ if $r - r_e \geq d_{ij}(x, y)$. However, this case study shows that the proposed sensor deployment algorithms do not depend on any specific type of sensor model. The sensor detection model can be viewed as a plug-in module when different types of sensor are encountered in applying the deployment algorithms.

24.4.5 Case Study 3

Next, we consider a terrain model with the existence of obstacles. We have manually placed one obstacle that occupies grid points $(7, 3), (7, 4)$, and another obstacle that occupies grid points $(3, 5), (4, 5), (5, 5)$. They are marked as "Obstacle" in Figure 24.3, which gives the layout of the setup for this case study. We have evaluated the proposed algorithms on the sensor detection model in case study 2, which is given by Equation (24.3) with the same model parameters as $r = 5, r_e = 4, \lambda = 0.5$, and $\beta = 0.5$. Figure 24.31 presents results for the truncated Gaussian models with different maximum errors. Figure 24.32 compares the coverage based on Equation (24.18) for MIN_MISS and MAX_MISS with coverage obtained without location uncertainty. It is obvious that, because of the existence of obstacles, the actual range of sensor detection is due to the line-of-sight principle. Therefore, the reduction in sensor detection range causes an increase in the number of sensors required for the same miss probability threshold, as shown in Figures 24.31 and 24.32.

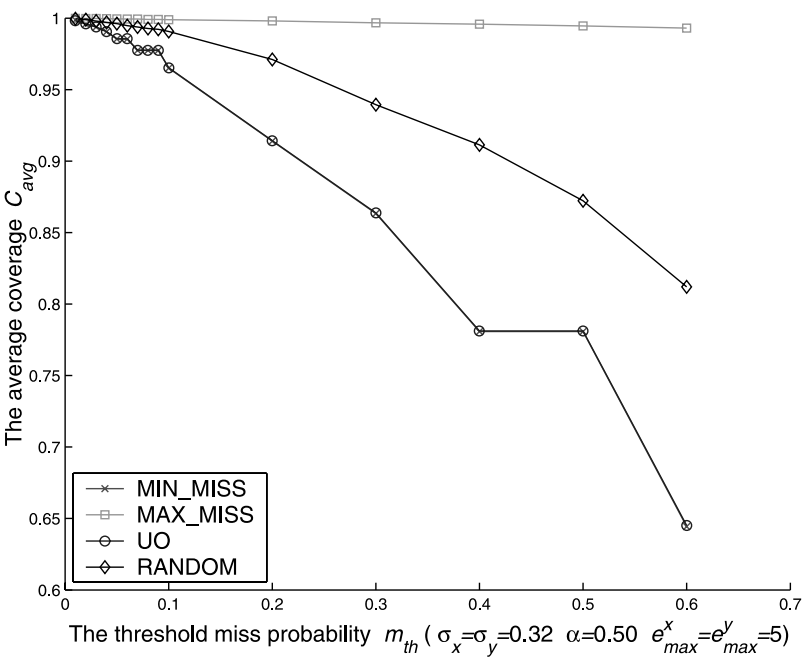


Figure 24.30. Comparison of average coverage for various placement schemes with $\alpha = 0.5, e^x_{\max} = e^y_{\max} = 5$, $\sigma_x = \sigma_y = 0.32$.

© 2005 by Chapman & Hall/CRC

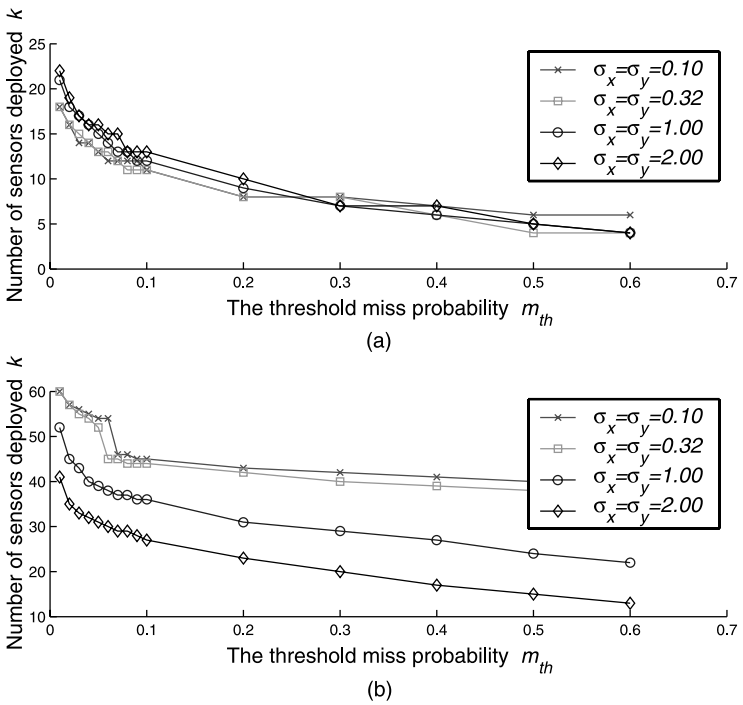


Figure 24.31. Number of sensors required as a function of the miss probability threshold in the presence of obstacles with $\alpha = 0.5, e^x_{\max} = e^y_{\max} = 5$ for and (a) MIN_MISS (b) MAX_MISS.

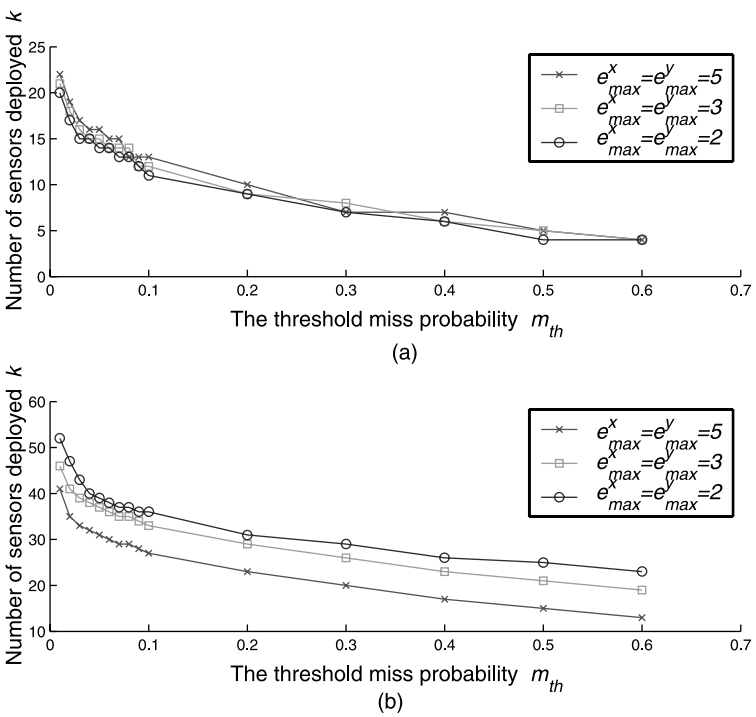


Figure 24.32. Comparisons of different truncated Gaussian models in the presence of obstacles with $\alpha = 0.5, \sigma_x = \sigma_y = 2$ for (a) MIN_MISS and (b) MAX_MISS.

© 2005 by Chapman & Hall/CRC

24.5 Conclusions

In this chapter we have discussed two important aspects in sensor node deployment for wireless sensor networks. The proposed VFA introduced in Section 24.3 improves the sensor field coverage considerably compared to random sensor placement. The sensor placement strategy is centralized at the cluster level, since every cluster head makes redeployment decisions for the nodes in its cluster. Nevertheless, the clusters make deployment decisions independently; hence, there is a considerable degree of decentralization in the overall sensor deployment. The virtual force in the VFA is calculated with a grid point being the location indicator and the distance between two grid points being a measure of distance. Furthermore, in our simulations, the preferential areas and the obstacles are both modeled as rectangles. The VFA, however, is also applicable for alternative location indicators, distance measures, and models of preferential areas and obstacles. Hence, the VFA can be easily extended to heterogeneous sensors, where sensors may differ from each other in their detection modalities and parameters.

In Section 24.4 we formulated an optimization problem on uncertainty-aware sensor placement. A minimum number of sensors are deployed to provide sufficient grid coverage of the sensor field, though the exact sensor locations are not known. The sensor location has been modeled as a random variable with a Gaussian probability distribution. We have presented two polynomial-time algorithms to optimize the number of sensors and determine their placement in an uncertainty-aware manner. The proposed algorithms address coverage optimization under constraints of imprecise detections and terrain properties.

Acknowledgments

The following are reprinted with permission of IEEE: Figures 24.1, 24.5–24.11, 24.13, 24.14, and 24.16 are taken from [26]. (c) 2003 IEEE; Figures 24.17 and 24.23–26 are taken from [27]. (c) 2003 IEEE; Figures 24.3, 24.18–24.21, and 24.27–24.32 are taken from [28]. (c) 2004 IEEE.

References

[1] Akyildiz, I.F. *et al.*, A survey on sensor networks, *IEEE Communications Magazine*, August, 102, 2002.

[2] Estrin, D. *et al.*, Next century challenges: Scalable coordination in sensor networks, in *Proceedings of IEEE/ACM MobiCom Conference*, 263, 1999.

[3] Pottie, G.J. and Kaiser, W.J., Wireless sensor networks, *Communications of the ACM*, 43, 51, 2000.

[4] Tilak, S. *et al.*, A taxonomy of wireless micro-sensor network models, *ACM Mobile Computing and Communications Review*, 6(2), 28, 2002.

[5] Agre, J. and Clare, L., An integrated architecture for cooperative sensing networks, *IEEE Computer*, 33, 106, 2000.

[6] Bulusu, N. *et al.*, GPS-less low-cost outdoor localization for very small devices, *IEEE Personal Communication Magazine*, 7(5), 28, 2000.

[7] Heidemann, J. and Bulusu, N., Using geospatial information in sensor networks, in *Proceedings of CSTB Workshop on Intersection of Geospatial Information and Information Technology*, 2001.

[8] Musman, S.A. *et al.*, Sensor planning for elusive targets, *Journal of Computer & Mathematical Modeling*, 25, 103, 1997.

[9] Clark, M.R. *et al.*, Coupled oscillator control of autonomous mobile robots, *Autonomous Robots*, 9(2), 189, 2000.

[10] Howard, A. *et al.*, Mobile sensor network deployment using potential field: a distributed scalable solution to the area coverage problem, in *Proceedings of 6th International Conference on Distributed Autonomous Robotic Systems*, 299, 2002.

© 2005 by Chapman & Hall/CRC

[11] Heo, N. and Varshney, P.K., A distributed self spreading algorithm for mobile wireless sensor networks, in *Proceedings of IEEE Wireless Communications and Networking Conference*, paper ID: TS48-4, 2003.
[12] Meguerdichian, S. *et al.*, Coverage problems in wireless *ad-hoc* sensor networks, in *Proceedings of IEEE Infocom Conference*, 3, 1380, 2001.
[13] Meguerdichian, S. *et al.*, Exposure in wireless *ad-hoc* sensor networks, in *Proceedings of Mobicom Conference*, July, 2001, 139.
[14] Chakrabarty, K. *et al.*, Grid coverage for surveillance and target location in distributed sensor networks, *IEEE Transactions on Computers*, 51, 1448, 2002.
[15] Chakrabarty, K. *et al.*, Coding theory framework for target location in distributed sensor networks, in *Proceedings of International Symposium on Information Technology: Coding and Computing*, 130, 2001.
[16] Dhillon, S.S. *et al.*, Sensor placement for grid coverage under imprecise detections, in *Proceedings of International Conference on Information Fusion*, 1581, 2002.
[17] Dhillon, S.S. and Chakrabarty, K., Sensor placement for effective coverage and surveillance in distributed sensor networks, in *Proceedings of IEEE Wireless Communications and Networking Conference*, paper ID: TS49-2, 2003.
[18] O'Rourke, J., *Art Gallery Theorems and Algorithms*, Oxford University Press, New York, NY, 1987.
[19] Bulusu, N. *et al.*, Adaptive beacon placement, in *Proceedings of the International Conference on Distributed Computing Systems*, 489, 2001.
[20] Kasetkasem, T. and Varshney, P.K., Communication structure planning for multisensor detection systems, in *Proceedings of IEE Conference on Radar, Sonar and Navigation*, 148, 2, 2001.
[21] Penny, D.E. The automatic management of multi-sensor systems, *in Proceedings of International Conference on Information Fusion*, 1998.
[22] Clouqueur, T. *et al.*, Sensor deployment strategy for target detection, in *Proceedings of 1st ACM International Workshop on Wireless Sensor Networks and Applications*, September, 42, 2002.
[23] Liu, J. *et al.*, A dual-space approach to tracking and sensor management in wireless sensor networks, in *Proceedings of 1st ACM International Workshop on Wireless Sensor Networks and Applications*, September, 2002, 131.
[24] Locateli, M. and Raber, U., Packing equal circles in a square: a deterministic global optimization approach, *Discrete Applied Mathematics*, 122, 139, 2002.
[25] Elfes, A. Occupancy grids: a stochastic spatial representation for active robot perception, in *Proceedings of 6th Conference on Uncertainty in AI*, 60, 1990.
[26] Zou, Y. and Chakrabarty, K., Sensor Deployment and Target Localization Based on Virtual Forces, *Proceedings of IEEE InfoCom*, 1293, 2003.
[27] Zou, U. and Chakrabarty, K., Uncertainty-aware sensor deployment algorithms for surveillance applications, *Proceedings of IEEE GlobeCom*, 2972, 2003.
[28] Zou, Y. and Chakrabarty, K., Uncertainty-aware and coverage-oriented deployment for sensor networks, *J. Parallel and Distributed Computing*, 64(7), 788, 2004.

© 2005 by Chapman & Hall/CRC

25

Deployment of Sensors: An Overview

S.S. Iyengar, Ankit Tandon, Qishi Wu, Eungchun Cho,
Nageswara S.V. Rao, and Vijay K. Vaishnavi

25.1 Introduction

25.1.1 What Is a Sensor Network?

A distributed sensor network (DSN) is a collection of a large number of heterogeneous intelligent sensors distributed logically, spatially, or geographically over an environment and connected through a high-speed network. The sensors may be cameras as vision sensors, microphones as audio sensors, ultrasonic sensors, infrared sensors, humidity sensors, light sensors, temperature sensors, pressure/force sensors, vibration sensors, radioactivity sensors, seismic sensors, etc. The sensors continuously monitor and collect measurements of respective data from their environment. The collected data are processed by an associated processing element that then transmits it through an interconnected communication network. The information that is gathered from all parts of the sensor network is then integrated using some data-fusion strategy. This integrated information is then useful to derive appropriate inferences about the environment where the sensors are deployed.

25.1.2 Example

With the emergence of high-speed networks and with their increased computational capability, DSNs have a wide range of real-time applications in aerospace, automation, defense, medical imaging, robotics, weather prediction, etc. To elucidate, let us consider sensors spread in a large geographical territory collecting data on various parameters, like temperature, atmospheric pressure, wind

© 2005 by Chapman & Hall/CRC

velocity, etc. The data from these sensors are not as useful when studied individually; but, when integrated, they give the picture of a large area. Changes in the data across time for the entire region can be used in predicting the weather at a particular location.

DSNs are a key part of the surveillance and reconnaissance infrastructure in modern battle spaces. DSNs offer several important benefits, such as ease of deployment, responsiveness to battlefield situations, survivability, agility, and easy sustainability. These benefits make DSNs a lethal weapon for any army, providing it with the high-quality surveillance and reconnaissance data necessary for any combat operation [1].

25.1.3 Computational Issues

Coordinated target detection, surveillance, and localization require efficient and optimal solutions to sensor deployment problems (SDPs), and have attracted a great deal of attention from several researchers. Sensors must be suitably deployed to achieve the maximum detection probability in a given region while keeping the cost within a specified budget. Recently, SDPs have been studied in a variety of contexts. In the adaptive beacon placement, the strategy is to place a large number of sensors and then shut some of them down based on their localization information. Most of the approaches are based on sensor devices with deterministic coverage capability. In reality, the sensor coverage is not only dependent on the geometric distance from the sensor, but also on other factors, such as environmental conditions and device noise. The deterministic models do not adequately capture the tradeoffs between sensor network reliability and cost. Thus, next-generation sensor networks must go beyond the deterministic coverage techniques to perform the assigned tasks, such as online tracking/monitoring in unstructured environments.

In reality, the probability of successful detection decreases in some way as the target moves further away from the sensor, because of less received power, more noise, and environmental interference. Therefore, the sensor detection is ''probabilistic.'' The sensor deployment is a complex task in DSNs because of factors such as different sensor types and detection ranges, sensor deployment and operational costs, and local and global coverage probabilities. Essentially, the sensor deployment is an optimization problem, which often belongs to the category of multi-dimensional and nonlinear problems with complicated constraints. When the deployment locations are restricted to (discrete) grid points, this problem becomes a combinatorial optimization problem but is still computationally very difficult. In particular, this problem contains a considerable number of local maxima, and it is very difficult for the conventional optimization methods to obtain its global maximum.

Distributed, real-time sensor networks are essential for effective surveillance in a digitized battlefield and environmental monitoring. There are several underlying challenges in the design of a sensor network. A key issue is the layout or distribution of sensors in the environment. The number, type, location, and density of sensors determine the layout of a sensor network. An intelligent placement of sensors can enhance the performance of the system significantly. Some redundancy is also needed for error detection and correction caused by faulty sensors and an unreliable communication network. At the same time, a large number of sensors correspond to higher deployment costs, the need of higher bandwidth, increased collisions in relaying messages, higher energy consumption, and more time-consuming algorithms for data fusion.

Usually, sensors are deployed in widespread hazardous, unreliable or possibly even adversarial environments, and it is essential that they do not require human attention very often. It is necessary that sensors are self-aware, self-configurable, autonomous, and self-powered. They must have enough energy reserves to work for a long period of time or they should be able to recharge themselves. Power in each sensor is finite and precious, and it is extremely essential to conserve it.

Sensors typically communicate through wireless networks, where bandwidth is significantly lower than the wired channels. Wireless networks are more unreliable and data-faulty; therefore, there is a need for robust, fault-tolerant routing and data-fusion algorithms. It is of the utmost importance to use

© 2005 by Chapman & Hall/CRC

techniques that increase the efficiency of data communication, thus reducing the number of overall bits transmitted and also reducing the number of unnecessary collisions. It has been found that, typically, it requires 100 to 1000 times more energy to transmit a bit than to execute an instruction, which means that it is beneficial to compress the data before transmitting it. Hence, it is essential to minimize data transfer in the sensor network to make it more energy efficient.

In real-time medical and military applications, it is sometimes essential to have an estimate of the message delay between two nodes of a sensor network. The current algorithms to compute sensor message delay are computationally very expensive and pose a challenge for further study.

25.2 Importance of Sensor Deployment

Sensor placement directly influences resource management and the type of back-end processing and exploitation that must be carried out with the sensed data in a DSN. A key challenge in sensor resource management is to determine a sensor field architecture that optimizes cost, and provides high sensor coverage, resilience to sensor failures, and appropriate computation/communication tradeoffs. Intelligent sensor placement facilitates unified design and operation of sensor/exploitation systems, and decreases the need for excessive network communication for surveillance, target location, and tracking. Therefore, sensor placement forms the essential ''glue'' between front-end sensing and back-end exploitation.

In a resource-bounded framework of a sensor network, it is essential to optimize the deployment of sensors and their transmission. Given a surveillance area, the most important challenge is to come up with the architecture of a ''minimalistic sensor network'' that requires the least number of sensors (with the lowest deployment costs) and has maximum coverage. It is also important that the sensors are deployed in such a manner that they transmit/report the minimum amount of sensed data. The ensemble of this data must contain sufficient information for the data-processing center to subsequently derive appropriate inferences and query a small set of sensors for detailed information.

In addition to the above, sensor networks must take into account the nature of the terrain of the environment where they would be deployed. In practical applications, sensors may be placed in a terrain that has obstacles, such as buildings and trees that block the line of vision of infrared sensors. Uneven surfaces and elevations of a hilly terrain may make communication impossible. In battlefields, radio jamming may make communication among sensors difficult and unreliable. Thus, while deploying the sensors, it is necessary to take the above factors into account and to estimate the need for redundancy of sensors due to the likelihood of sensor failures, and the extra power needed to transmit between deployed sensors and cluster heads.

In the case of mobile sensors, the sensor fields are constructed such that each sensor is repelled by both obstacles and by other sensors, thereby forcing the network to spread itself through the environment. However, most of the practical applications, like environmental monitoring, require static sensors, and the above scenario of self-deployment does not provide a solution.

Some applications of sensor networks require target detection and localization. In such cases, deployment of sensors is the key aspect. In order to achieve target localization in a given area, the sensors have to be placed in such a manner that each point is sensed by a unique set of sensors. Using the set of sensors that sense the target, an algorithm can predict or pinpoint the location of the target.

The above issues clearly prove that sensor placement is one of the most key aspects of any DSN architecture, and efficient algorithms for computing the best layout of sensors in a given area need to be researched. Using the concept of Eisenstein integers, one such algorithm that computes efficient placement of sensors of a distributed sensor network covering a bounded region on the plane is presented next. The number of sensors required in the distributed sensor network based on Eisenstein integers is about $4/3\sqrt{3} \approx 0.77$ of the number of the sensors required by the traditional rectangular grid-point-based networks covering the same amount of area.

© 2005 by Chapman & Hall/CRC

25.3 Placement of Sensors in a DSN Using Eisenstein Integers

25.3.1 Introduction

A DSN covering a region in the plane $\mathbb{R}^2$ such that each lattice point (grid point) can be detected by a unique set of responding sensors is convenient for locating stationary or mobile targets in the region. In such sensor networks, each set of responding sensors uniquely identifies a grid point corresponding to the location of the target [2]. Moreover, the location of the target is easily computed from the set of responding sensors, the locations of which are fixed and known. For simplicity, we assume that both the sensors and the targets are located only on lattice points. More realistically, we may require only sensors to be placed at lattice points and targets are located by finding the nearest lattice points. We consider the ring of Eisenstein integers, which have direct applications to the design of a DSN.

25.3.2 Eisenstein Integers

Gaussian integers are complex numbers of the form $a + bi$, where a and b are integers. Gaussian integers form a standard regular rectangular grid on a plane, which we will call a Gaussian grid. Let G be the set of Gaussian integers

$$G = \{a + b\mathrm{i} : a, b \in \mathbb{Z}\}$$

G is closed under addition, subtraction, and multiplication.

$$(a + b\mathrm{i}) \pm (c + d\mathrm{i}) = (a \pm c) + (b \pm d)\mathrm{i}$$
$$(a + b\mathrm{i})(c + d\mathrm{i}) = (ac - bd) + (ad + bc)\mathrm{i}$$

In other words, G is invariant under the addition (translation) and multiplication (dilation) by any Gaussian integer and G is a subring of $\mathbb{C}$, the field of complex numbers. We may consider any point (x, y) in the two-dimensional real plane $\mathbb{R}^2$ as a complex number $x + \mathrm{i}y$, i.e. as a point in the complex plane $\mathbb{C}$. G is the set of all integer lattice points of $\mathbb{R}^2$.

Recall that i is the primary fourth root of 1, i.e. $\mathrm{i}^4 = 1$ and any complex number z with $z^4 = 1$ is i^k for some $k \in \mathbb{Z}$. In fact, z is either 1, -1, i or $-\mathrm{i}$. If i is replaced by ω, the primary third root of 1, then we get Eisenstein integers. The primary root of ω is of the form

$$\mathrm{e}^{2\pi i/3} = \cos 2\pi/3 + \mathrm{i} \sin 2\pi/3 = -1/2 + \sqrt{3}/2\mathrm{i}$$

and satisfies $\omega^2 + \omega + 1 = 0$. This means that any integer power of ω can be represented as a linear combination of 1 and ω.
Let E be the set of Eisenstein integers

$$E = \{a + b\omega : a, b \in \mathbb{Z}\}$$

E is also invariant under the translation and dilation by any Eisenstein integer and E forms a subring of $\mathbb{C}$, since

$$(a + b\omega) \pm (c + d\omega) = (a \pm c) + (b \pm d)\omega$$
$$(a + b\omega)(c + d\omega) = (ac - bd) + (ad + bc - bd)\omega$$

The three solutions of $z^3 = 1$, given by 1, ω and $\omega^2 = -1 - \omega$ form an equilateral triangle. The Eisenstein integers ± 1, $\pm\omega$, $\pm(1+\omega)$ are called the Eisenstein units. Eisenstein units form a regular hexagon centered at the origin [3]. As G yields a tessellation of $\mathbb{R}^2$ by squares, E forms a tessellation of $\mathbb{R}^2$ by equilateral triangles and its dual forms a tessellation of $\mathbb{R}^2$ by regular hexagons. The main theorem of this paper is as follows. A distributed sensor network whose sensors (with unit range) are

© 2005 by Chapman & Hall/CRC

placed at Eisenstein integers of the form $m + n\omega$ with $m + n \equiv 0 \bmod 3$ detects the target on Eisenstein integers uniquely. Each location at an Eisenstein integer $a + b\omega$ is detected by one sensor located at itself, by the set of three sensors placed at $\{(a+1) + b\omega, a + (b+1)\omega, (a-1) + (b-1)\omega\}$, or by the set of three sensors placed at $\{(a-1) + b\omega, (a+1) + (b+1)\omega, a + (b-1)\omega\}$. In practical applications, the location of the target is easily approximated either by the location of the sensor itself (if there is only one responding sensor) or simply the average $(a_1 + a_2 + a_3)/3 + (b_1 + b_2 + b_3)\omega/3$ of the three Eisenstein integers $a_i + b_i\omega$. The proof of the theorem will be given after more mathematical background on Eisenstein integers and tessellation is given.

Six equilateral triangles sharing a common vertex form a regular hexagon, which generates a hexagonal tessellation of $\mathbb{R}^2$. E is the subring of $\mathbb{C}$, which means E an additive subgroup of $\mathbb{C}$, is closed under complex multiplication satisfying the usual associative, commutative, and distributive properties. ω generates a multiplicative subgroup $\{1, \omega, \omega^2\}$ of the circle, called a cyclotomic (circle cutting) subgroup of order 3. Eisenstein units $1, -\omega^2, \omega, -1, \omega^2, -\omega$ form a cyclotomic subgroup of order 6 (and a regular hexagon centered at the origin).

Each closed unit disk centered at a Gaussian integer $m + ni$ contains four other Gaussian integers $(m \pm 1) + n\omega$, $m + (n \pm 1)\omega$, and $(m-1) + (n \pm 1)\omega$. Any point in $\mathbb{R}^2$ is within a $1/\sqrt{2}$ radius of a Gaussian integer and within a $1/\sqrt{3}$ radius of an Eisenstein integer. Let $N(e)$ be the neighborhood of $e \in E$ in $\mathbb{R}^2$, defined as the set of all points for which the closest point in E is e, i.e. the set of all points which are not farther from e than from any other points in E

$$N(e) = \{x \in \mathbb{R}^2 : ||x - e|| \,\Omega\, ||x - f|| \forall f \in E\}$$

$N(e)$ is the regular hexagon centered at e with the edge length $1/\sqrt{3}$ whose vertices are the centers of equilateral triangles of Eisenstein tessellation, and the area of $N(e)$ is $\sqrt{3}/2$. The regular hexagons $N(e)$ for $e \in E$ form a tessellation of $\mathbb{R}^2$. Each N(e) contains exactly one Eisenstein integer (namely, e). In this sense, the density of E in $\mathbb{R}^2$ is $2/\sqrt{3}$, the inverse of the area of $N(e)$. A similar argument shows $N(g)$, the set of all points in $\mathbb{R}^2$ for which the closest point in G is g, is a square centered at g with unit side whose vertices are centers of the Gaussian square tessellation. The density of Gaussian integers G is unity, which is lower than the density of E.

25.3.3 Main Theorem

Now we consider a DSN covering the complex plane such that each Eisenstein integer (grid point) can be detected by a unique set of responding sensors. That is, a distributed sensor network with the property that for each set of responding sensors there is a unique Eisenstein integer corresponding to the location of the target. Moreover, the location of the target is easily computed from the set of responding sensors that are fixed and known points at Eisenstein integers.

A DSN whose sensors (with unit range) are placed at Eisenstein integers of the form $m + n\omega$ with $m + n \equiv 0 \bmod 3$ detects each Eisenstein integer uniquely. Each Eisenstein integer $a + b\omega$ is detected by one sensor located at itself, by a set of three sensors placed at $\{(a+1) + b\omega, a + (b+1)\omega, (a-1) + (b-1)\omega\}$, or by the set of three sensors placed at $\{(a-1) + b\omega, (a+1) + (b+1)\omega, a + (b-1)\omega\}$.

Proof. The minimum distance between distinct points in E is unity and a sensor placed at a point $e = a + b\omega \in E$ detects six neighbor points in E in addition to itself. The six neighbors are $(a \pm 1) + b\omega$, $a + (b \pm 1)\omega$, and $(a-1) + (b \pm 1)\omega$, which form a regular hexagon centered at e. Consider the hexagonal tessellation of $\mathbb{R}^2$ generated by the regular unit hexagon with vertices at $\pm 1, \pm\omega$, and $-1 \pm \omega$ with center at $e = 0$, the origin of the complex plane. Let V be the set of all vertices of the tessellation and C be the set of all centers of the hexagons of the tessellation. We note $E = V \cup C$ and $V \cap C = \emptyset$, i.e. every Eisenstein integer is either a vertex of the tessellation or the center of the hexagons. The minimum distance between distinct points in C is $1/\sqrt{3}$ and every point in C is of the form $e = a + b\omega$ with $a + b \equiv 0 \bmod 3$. For example, $0, 1 + 2\omega, 2 + \omega, 1 - \omega, \ldots$. For each v in V, there exist exactly three points c_1, c_2 and c_3 in C such that $\text{dist}(v, c_i) = 1$ and $(c_1 + c_2 + c_3)/3$, with $\text{dist}(v, c_i) = 1$. This means

© 2005 by Chapman & Hall/CRC

that if the sensors are placed at the points in C, the centers of the hexagons tessellating the plane, then every point e in E is detected either by a single sensor (when e belongs to C) or by a set of three sensors (when e belongs to V).

Remark. Hexagonal tessellation is the most efficient tessellation (there are only two more tessellations of a plane by regular polygons: square tessellation and triangular tessellation) in the sense that the vertices belong to exactly three neighboring hexagons (square tessellation requires four and triangular tessellation six) and each set of three neighboring hexagons has only one vertex in common.

25.3.4 Conclusion

In practical applications, the location of a target is easily approximated with such sensor networks. Assuming the targets are located on grid points only, the target location is either the position of the sensor itself (if there is only one responding sensor), or simply the average $(a_1 + a_2 + a_3)/3 + (\mathrm{b}_1 + \mathrm{b}_2 + b_3)\omega/3$ of the three Eisenstein integers $a_i + b_i\omega$. More generally, the target location is approximated either by the position of the sensor itself (if there is only one responding sensor), or by the average $(a_1 + a_2)/2 + (b_1 + b_2)\omega/2$ of the two Eisenstein integers (if there are two responding sensors) or the average $(a_1 + a_2 + a_3)/3 + (b_1 + b_2 + b_3)\omega/3$ of the three Eisenstein integers $a_i + b_i\omega$ (if there are three responding sensors).

A similar result follows for the sensor network based on a Gaussian lattice whose sensors are placed at Gaussian integers $a + b\mathrm{i}$, where $a + b \equiv 0 \bmod 2$. The minimum distance between sensors in this network is $\sqrt{2}$. A target at a Gaussian integer $a + b\mathrm{i}$ with $a + b \equiv 0 \bmod 2$ is detected by the sensor placed on it. Otherwise, that is $a + b \equiv 1 \bmod 2$, the target is detected by four sensors placed at the four neighboring Gaussian integers $(a \pm 1) + b\mathrm{i}$ and $a + (b \pm 1)\mathrm{i}$. The average density of the sensors in the Gaussian-integer-based network is about $\frac{1}{2}$, whereas the average density for the network based on the Eisenstein integer is about $2/3\sqrt{3} \approx 0.38$. In other words, the Eisenstein network requires less sensors (about $4/3\sqrt{3} \approx 0.77$) that the former.

25.4 Complexity Analysis of Efficient Placement of Sensors on Planar Grid

One of the essential tasks in the design of distributed sensor systems is the deployment of sensors for an optimal surveillance of a target region while ensuring robustness and reliability. Those sensors with probabilistic detection capabilities with different costs are considered here. An SDP for a planar grid region is formulated as a combinatorial optimization problem to maximize the overall detection probability within a given deployment cost. This sensor placement problem is shown to be NP-complete, and an approximate solution is proposed based on the genetic algorithm method. The solution is obtained by the specific choices of genetic encoding, fitness function, and genetic operators (such as crossover, mutation, translocation, etc.) for this problem. Simulation results are presented to show the benefits of this method, as well as its comparative performance with a greedy sensor placement method.

25.4.1 Introduction

Sensor deployment is important for many strategic applications, such as coordinated target detection, surveillance, and localization. There is a need for efficient and optimal solutions to these problems. Two different, but related, aspects of sensor deployment are the target detection and localization. For optimal detection, sensors must be suitably deployed to achieve the maximum detection probability in a given region while keeping the cost within a specified budget. To localize a target inside the surveillance

© 2005 by Chapman & Hall/CRC

region, the sensors must be strategically placed such that every point in the surveillance region is covered by a unique subset of sensors [4,5]. The research work presented in this paper is focused on the first aspect.

Optimal SDP have been studied in a variety of contexts. Recently, in adaptive beacon placement, the strategy is to place a large number of sensors and then shut some of them down based on their localization information. In this context, Bulusu and *co-workers* [6,7] consider the evaluations for spatial localization based on radio-frequency proximity, and present an adaptive algorithm based on measurements. In a related area, Guibas *et al.* [8] present a unique solution to the visibility-based pursuit evasion problem in robotics applications. In this context, Meguerdichian *et al.* [9] describe coverage problems in wireless *ad hoc* sensor networks given the global knowledge of node positions, using a Voronoi diagram for maximal breach path for worst-case coverage and Delaunay triangulation for maximal support paths for best-case coverage. These approaches are based on sensor devices with deterministic coverage capability. In practice, the sensor coverage is not only dependent on the geometrical distance from the sensor [9], but also on factors such as environmental conditions and device noise. As such, the deterministic models do not adequately capture the tradeoffs between sensor network reliability and cost. Thus, the next-generation sensor networks must go beyond the deterministic coverage techniques to perform the assigned tasks, such as online tracking/monitoring in unstructured environments.

In practical sensors, the probability of successful detection decreases as the target moves further away from the sensor, because of less received power, more noise, and environmental interference. Therefore, the sensor detection is ''probabilistic,'' which is the focus of this paper. The sensor deployment is a complex task in DSNs because of factors such as different sensor types and detection ranges, sensor deployment and operational costs, and local and global coverage probabilities [10,11]. Essentially, the sensor deployment is an optimization problem, which often belongs to the category of multi-dimensional and nonlinear problems with complicated constraints. If the deployment locations are restricted to discrete grid points, then this problem becomes a combinatorial optimization problem, but it is still computationally very difficult. In particular, this problem contains a considerable number of local maxima, and it is very difficult for the conventional optimization methods to obtain its global maximum [12].

A generic SDP over the planar grid to capture a subclass of sensor network problems can now be formulated. Consider sensors of different types, wherein each type is characterized by a detection region and an associated detection probability distribution. Thus, each deployed sensor detects a target located in its region with certain probability and incurs certain cost. Also, consider an SDP that deals with placing the sensors at various grid points to maximize the probability of detection while keeping the cost within a specified limit.

In this section, it is shown that this sensor deployment problem is NP-complete, and hence it is unlikely that one will find a polynomial-time algorithm for solving it exactly. Next, an approximate solution to this problem using the genetic algorithm [13] for the case where the sensor detection distributions are statistically independent is presented. The solution presented is based on specifying the components of the genetic algorithm to suit the SDP. In particular, the genetic encoding and fitness function is specified to match the optimization criterion, and also specify the crossover, mutation, and translocation operators to facilitate the search for the near-optimal solutions. In practice, near-optimality is often good enough for this class of problems. Simulation results are then presented for 50×50 or larger grids with five or more sensor types when the *a priori* distribution of target is uniform. The solution proposed is quite effective in yielding solutions with good detection probability and low cost. A comparison of the proposed method with a greedy approach of uniformly placing the sensors over the grid follows next. From the comparison, it is found that this method achieved significantly better target detection probability within the budget.

The rest of this text is organized as follows. In Section 25.4.2, a formulation of the sensor deployment problem is given and it is shown to be NP-complete. In Section 25.4.3, an approximate solution using a genetic algorithm is presented. Section 25.4.4 discusses the experimental results.

© 2005 by Chapman & Hall/CRC

25.4.2 The SDP

In this section, the SDP is formulated, and then it is shown to be NP-complete.

25.4.2.1 Surveillance Region

A planar surveillance region R is to be monitored by a set of sensors to detect a target T if located somewhere in the region (our overall method is applicable to three dimensions). The planar surveillance region is divided into a number of uniform contiguous rectangular cells with identical dimensions, as shown in Figure 25.1. Each cell of R is indexed by a pair (i, j), and $C(i, j)$ denotes the corresponding cell. Let l_x and l_y denote the dimensions of a cell along the x and y coordinates respectively. As Figure 25.1 shows, a circular coverage area is approximated by a set of cells within a certain maximum detection distance of sensor S_k.[2] When the ratio of sensor detection range to cell dimension is very large, the sensor coverage area made up of many tiny rectangular cells will approach the circle.

There are q types of sensor and a sensor of the kth type is denoted by S_k for $k \in \{1, 2, \ldots, q\}$. There are N_k sensors of type k. A sensor S can be deployed in the middle of $C(i, j)$ to cover the discretized circular area $A_S(i, j)$ consisting of cells as shown in Figure 25.1. A sensor S_k deployed at cell (i, j) detects the target $T \in A_{S_k}(i, j)$ according to the probability distribution $P\{S_k | T \in A_{S_k}(i, j)\}$ while incurring the cost $w(k)$.

A *sensor deployment* is a function $\Re$ from the cells of R to $\{\varepsilon, 1, 2, \ldots, q\}$ such that $\Re(i, j)$ is the type of sensor deployed at the cell (i, j); $\Re(i, j) = \varepsilon$ indicates no sensor is deployed, i.e. $w(\varepsilon) = 0$. The *cost of a sensor deployment* $\Re$ is the sum of cost of all sensors deployed in region R, given by

$$\text{Cost}(\Re) = \sum_{C(i,j) \in R} w(\Re(i, j))$$

The *detection probability* of deployment $\Re$, given by $P\{\Re | T \in R\}$, is the probability that a target T located somewhere in region R will be detected by at least one deployed sensor. The SDP considered in this paper can now be formally stated:

> Given a surveillance region R, cost budget Q, q types of sensor, and N_k sensors of type k, find a sensor deployment $\Re$ to maximize detection probability $P\{\Re | T \in R\}$ under the constraint $\text{Cost}(\Re) \leq Q$.

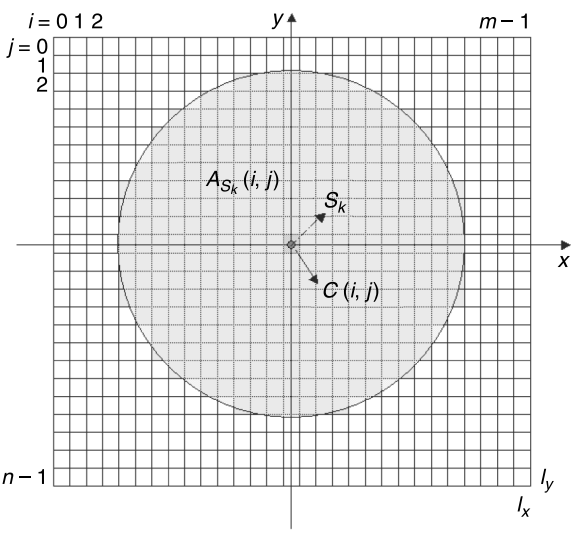


Figure 25.1. Surveillance region R is divided into $m\,n$ rectangular cells, with sensor S_k deployed in cell $C(i, j)$ covering area $A_{S_k}(i, j)$ in a probabilistic sense.

[2]In Figure 25.1, a cell is shaded if and only if its center is located within the sensor's maximum detection range.

© 2005 by Chapman & Hall/CRC

Informally, it is required to locate the sensors of various types on the grid points to achieve the maximum detection probability while keeping the deployment cost within a specified budget. The *decision version* of SDP asks for a deployment with detection probability at least A under the same cost condition, i.e. $P\{\Re|T \in R\} \geq A$ and $\text{Cost}(\Re) \leq Q$. The traditional polygon or rectangle coverage problems, studied in VLSI and related areas, focus on covering regions with a minimum number of circles or rectangles and do not incorporate the probabilistic aspects of the sensors [14].

25.4.2.2 Sensor Detection Distributions

Now, one can briefly describe some detection probability distributions used in SDP. The exact form of the distributions is not critical to the discussion in this text, only that they be in computable form.

Consider that each sensor type is specified by its local detection probability of detecting a target at a point within its detection region. With regard to a sensor, detection is more likely as a target approaches the sensor. The cumulative detection probability of a sensor for a region is computed by integrating its local detection probability for detecting a target as the target gets close to the sensor, passes near the sensor, and then leaves it behind. In general, there are two ways of modeling sensor detection performance, based on how the integrated detection probability is approximated [15].

- *Definite Range Law Approximation* (*Cookie Cutter*). In this model, only one parameter, i.e. maximum detection range, is used. A target is always detected if it lies within a certain distance of the sensor, or it is never detected if it lies beyond the sensor's maximum detection range, as Figure 25.2 shows.
- *Imperfect Sensor Approximation*. Besides the maximum detection range, a second parameter, mean detection probability (less than one), is specified for such a sensor model, as Figure 25.3 shows.

Comparing the above two approximations, we suggest that the latter models the real situations more reasonably. Based on the imperfect sensor approximation, a more accurate sensor performance model may be specified by a Gaussian cumulative detection probability instead of mean detection probability to approximate a real sensor.

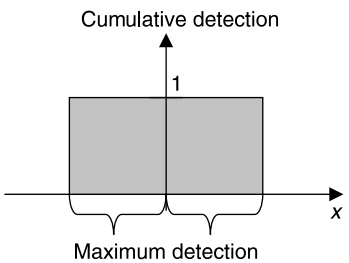


Figure 25.2. Definite range law.

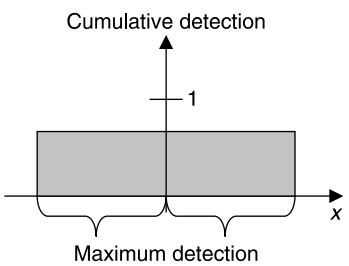


Figure 25.3. Imperfect sensor.

© 2005 by Chapman & Hall/CRC

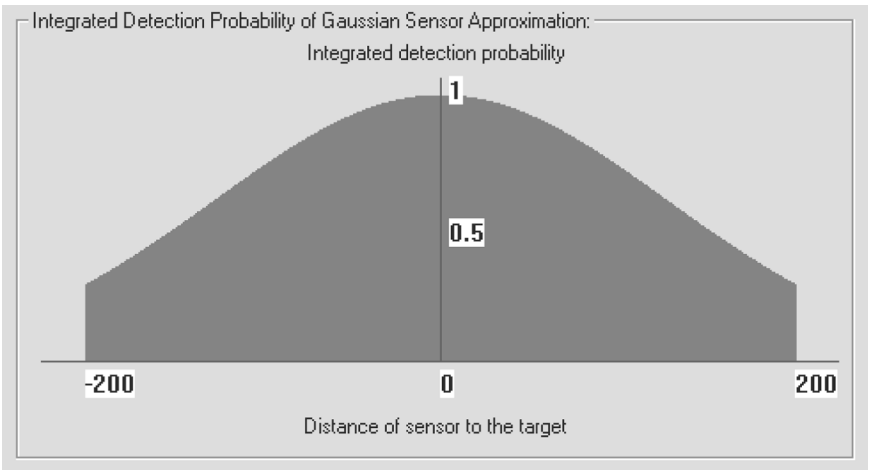


Figure 25.4. Integrated detection probability of Gaussian sensor.

Given the detection probability density function $p_{S_k}(x)$ for a sensor of type k, the detection probability $P\{S_k|T \in C(i,j)\}$ for cell $C(i,j)$ is given by

$$P\{S_k|T \in C(i,j)\} = \int_{x \in C(i,j)} p_{S_k}(x)\,\mathrm{d}x$$

After obtaining all individual detection probabilities for the cells covered by sensor S_k, Gaussian function to compute the cumulative detection probability may be used. In this paper, imperfect sensor approximation with a Gaussian cumulative detection probability to abstract a real sensor is used. Consider

$$P(S_k, \tau, \alpha_{S_k}) = P\{S_k, \tau, \alpha_{S_k}|T \in A_{S_k,\tau}\} = \mathrm{e}^{\tau^2/2\alpha_{S_k}^2} \quad \tau \in [0, \mathrm{d}_{S_k}]$$

which is the detection probability for a target located at distance τ from the sensor. The sensor detection quality coefficient α_{S_k} determines the shape of the detection probability curve. Distance τ is in the range between zero and the maximum detection distance d_{S_k}. A typical integrated detection probability of Gaussian sensor approximation is shown in Figure 25.4, where the measure of detection probability is assumed to reach unity when the target is very close to the sensor. We utilize this distribution in our computations for detection probability, but the genetic algorithm method presented here is applicable to suitably computable sensor distributions.

25.4.2.3 NP-Completeness of Sensor Deployment Problem

The sensor deployment problem can now be shown to be NP-complete by reducing the Knapsack problem (KP) to a special case of SDP, wherein each sensor monitors a single cell with a specified probability. Consider q-KP. Given a set U of n items such that, for each $u \in U$, we have size $s(u) \in \mathbb{Z}^+$ and the value $v(u) \in \mathbb{Z}^+$, does there exist a subset $V \in U$ of exactly q items such that $\sum_{u \in V} s(u) \leq B$ and $\sum_{u \in V} v(u) \geq K$ for given B and K? Note that exactly q items are required, as opposed to an unrestricted value in the usual KP; note that KP and q-KP are polynomially equivalent [16], since $q \leq n$ and the input for either problem instance has at least n items.

Consider the decision version of the SDP that asks for exactly q sensors to be deployed. Reduce the q-KP to a particular restriction of the SDP, denoted by q-SDP, such that only one sensor of each type is given, and each sensor S monitors a single cell and when two sensors are located in the same cell only one of them detects the target (i.e. suitable conditional probabilities are zero). For this special case, to maximize the detection probability, without the loss of generality, each cell is assumed to be occupied

© 2005 by Chapman & Hall/CRC

by no more than one sensor. Furthermore, under the uniform prior distribution of target T in cells combined with the nonoverlapping sensor regions, the probability of detection is simply the average of the probability of detection of the sensors deployed. Considering the sensor deployment $\Re$ deploys q sensors, the equation now becomes

$$P\{\Re|T \in R\} = \frac{1}{q} \sum_{\Re(i,j)=k;\Re(i,j)\neq\varepsilon} P\{S_k|T \in A_{S_k}\}$$

Given an instance of q-KP, each $u \in U$ can be mapped to a sensor S_u such that its cost and value are given by $w(u) = s(u)$ and

$$P\{S_u|T \in A_{S_u}\} = \frac{v(u)}{\sum_{a\in U} v(a)}$$

Then specify the sensor cost bound as $Q = B$ and the detection probability as

$$A = \frac{K}{q\sum_{a\in U} v(a)}$$

Given a solution to q-KP, a solution to q-SDP exists by just placing the sensors corresponding to the members of V on nonoverlapping grid points. Let (i_u, j_u) be the cell receiving a sensor due to $u \in V$. Then

$$\sum_{\Re(i,j)\neq\varepsilon;\Re(i,j)=k} w(k) = \sum_{\Re(i_u,j_u):u\in V} w(k) = \sum_{u\in V} s(u) \leq Q$$

which satisfies the first condition for q-SDP. For the second condition

$$P\{\Re|T \in R\} = \frac{1}{q} \sum_{\Re(i,j)=k;\Re(i,j)\neq\varepsilon} P\{S_k|T \in A_{S_k}\} = \frac{1}{q} \sum_{\Re(i_u,j_u)=k;\Re(i_u,j_u)\neq\varepsilon} \frac{v(u)}{\sum_{a\in U} v(a)} \geq A$$

Given a solution to the SDP, the solutions to q-KP can be obtained by choosing the items corresponding to the sensors deployed. Let $u_{(i,j)}$ denote the chosen item in corresponding to sensor located at $\Re(i,j)$. The first condition for q-KP follows from

$$\sum_{u\in V} s(u) = \sum_{u_{(i,j)}} s(u_{(i,j)}) = \sum_{\Re(i,j)=k} w(k) \leq B$$

The second condition for q-KP follows from

$$\sum_{u\in V} v(u) = \sum_{u_{(i,j)}} v(u_{(i,j)}) = \sum_{a\in U} v(a) \sum_{\Re(i,j)=k;\Re(i,j)\neq\varepsilon} P\{S_k|T \in A_{S_k}\}$$

$$= q\sum_{a\in U} v(a)P\{\Re|T \in R\} \geq Aq\sum_{a\in U} v(a) = K$$

It has been shown that SDP is NP-complete even when severe restrictions are imposed on the joint distributions, which is an indication of the computational complexity of this problem. Thus, it is unlikely that polynomial-time solutions that optimally solve SDP exist, which is a motivation to consider approximate solutions.

© 2005 by Chapman & Hall/CRC

25.4.2.4 Sensor Detection Probability Under Independence Condition

In this section, a restricted version of the SDP is considered, such that the sensors satisfy a certain statistical independence condition, which enables the joint detection probabilities to be computed efficiently. To guarantee high probability of detection, sensor detection range should overlap to insure that the critical areas of the surveillance region are covered by at least one sensor [17]. The local detection probability $P\{\Re|T \in C(i,j)\}$ must be suitably accumulated for each cell $C(i,j)$ covered by two or more sensors. To determine the sensor detection probabilities for such cells, we consider a simplest case first with two detection probabilities: $P\{S_m|T \in C(i,j)\}$ and $P\{S_n|T \in C(i,j)\}$, corresponding to sensors S_m and S_n which overlap in $C(i,j)$. The detection probability $P\{S_m \vee S_n|T \in C(i,j)\}$ is the probability of detecting the target successfully by at least one of the two sensors. Let $P(S_l)$ denote $P\{S_l|T \in C(i,j)\}$, for $l = m, n$, and let $P(S_m \vee S_n)$ denote $P\{S_m \vee S_n|T \in C(i,j)\}$. There are two mutually exclusive and collectively exhaustive cases for the successful detection, as Figure 25.5 shows.

Assuming that sensors S_m and S_n are *statistically independent*, such that

$$P(S_m \wedge S_n) = P(S_m)P(S_n)$$

we have

$$P(S_m \vee S_n) = P(S_m) + P(S_n) - P(S_m)P(S_n)$$

For a general case of n sensors covering a cell, by the inclusion–exclusion principle [18] we have:

$$\begin{aligned} P(S_1 \vee S_2 \vee \cdots \vee S_n) &= P(S_1 \vee S_2 \vee \cdots \vee S_{n-1}) + P(S_n) - P(S_1 \vee S_2 \vee \cdots \vee S_{n-1})P(S_n) \\ &\cdots \\ &= \sum_{i=1}^{n} P(S_i) - \sum_{1 \le i < j \le n} P(S_i)P(S_j) \\ &\quad + \sum_{1 \le i < j < k \le n} P(S_i)P(S_j)P(S_k) + \cdots + (-1)^{n-1} P(S_1)P(S_2)\cdots P(S_n) \end{aligned} \tag{25.1}$$

The overlap of local detection probabilities for n sensors is computed by applying the simple formula in Equation (25.1) repeatedly for each additional sensor as follows for computing $P\{\Re|T \in C(i,j)\}$ for each cell $C(i,j)$:

Step 1: Initialize local coverage probabilities and total cost.
Step 2: Locate a cell in which a sensor is deployed.
Step 3: Determine the sensor type.
Step 4: Update total cost.
Step 5: Compute the detection area of this sensor using Equation (25.1).

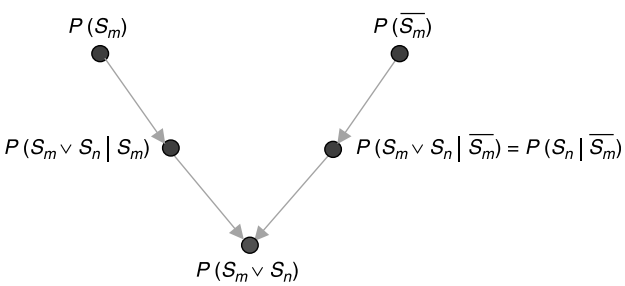


Figure 25.5. Multiple cases to successful detection in the simplest case.

© 2005 by Chapman & Hall/CRC

Step 6: For each of the cells within the discretized circular detection area, compute the overlapping detection probability.

Step 7: Update local detection probability for each cell covered by this sensor.

Step 8: Go back to Step 2 until all cells in the whole surveillance region are examined.

The details of the algorithm to compute the local coverage probabilities and total cost outlined above are presented as follows:

Input: sensor deployment scheme $\Re$

Output: local coverage probability $P\{\Re|T \in C(i,j)\}$ for each cell $C(i,j)$ and total cost, where $i = 0, 1, 2, \ldots, m-1$, $j = 0, 1, 2, \ldots, n-1$

Begin
  Initialize $P\{\Re|T \in C(i,j)\}$ *to* 0;
  Initialize $\text{Cost}(\Re)$ *to* 0;
  for $i = 0$ *to* $m-1$
  { *for* $j = 0$ *to* $n-1$
    { *let* $k = \Re(i,j)$;
      if $(k == \varepsilon)$
        continue
      else
        Update(k)
    }
  }
End

Auxiliary function *Update* (*Sensor k*)

Update(*Sensor k*)
Begin
  let $\text{Cost}(\Re) = \text{Cost}(\Re) + w(S_k)$;
  let $a = \left\lfloor \frac{d_{S_k}}{l_x} \right\rfloor$;
  let $b = \left\lfloor \frac{d_{S_k}}{l_y} \right\rfloor$;
  for $r = -a$ *to* a
    for $s = -b$ *to* b
    { *let* $\tau = \sqrt{(rl_x)^2 + (sl_y)^2}$;
      if $(\tau \le d_{S_k})$
      {
        let $overlap = P\{\Re|T \in C(i+r, j+s)\} * P(S_k, \tau, \alpha_{S_k})$;
        let $P\{\Re|T \in C(i+r, j+s)\} = P\{\Re|T \in C(i+r, j+s)\} + P(S_k, \tau, \alpha_{S_k})) - overlap$;
      }
    }
End

© 2005 by Chapman & Hall/CRC

Then compute $P\{\Re|T \in R\}$ by adding all local detection probabilities in the surveillance region given by

$$P\{\Re|T \in R\} = \sum_{i=0}^{m-1}\sum_{j=0}^{n-1} P\{\Re|T \in C(i,j)\} * P\{T \in C(i,j)\} \tag{25.2}$$

which is the objective function to be maximized under the cost condition $\text{Cost}(\Re) \leq Q$. Given the *a priori* distribution $P\{T \in C(i,j)\}$ of target T in a computable form and the sensor distributions, the objective function can be computed.

This version of the SDP, namely under the statistical independence condition, can be shown to be NP-complete by a simple extension of the results of the last section. Under the statistical independence, within each cell the probability of joint detection is the product of individual probabilities, and hence is smaller than either. Thus, any overlapping sensors within a cell can be separated to increase the probability of detection, and it suffices to consider no more than one sensor per cell. The rest of the proof follows the last section: under the restriction that each sensor detects a target in the cell it is currently located, this problem reduces to q-SDP in the last section, which shows the current problem to be NP-complete by restriction.

25.4.3 Genetic Algorithm Solution

A genetic algorithm is a computational model that simulates the process of genetic selection and natural elimination in biological evolution [19]. It has been frequently used to solve combinatorial and nonlinear optimization problems with complicated constraints or nondifferentiable objective functions [20,21]. The computation of a genetic algorithm is an iterative process towards achieving global optimality. During the iterations, candidate solutions are retained and ranked according to their quality. A fitness value is used to screen out unqualified solutions. Genetic operations of crossover, mutation, translocation, inversion, addition, and deletion are then performed on those qualified solutions to create new candidate solutions of the next generation. The above process is carried out repeatedly until a certain stopping or convergence condition is met. For simplicity, a maximum number of iterations can be chosen to be the stopping condition. The variation difference of the fitness values between two adjacent generations may also serve as a good indication for convergence. To utilize the genetic algorithm method, various parts of the SDP must be mapped to the components of the genetic algorithm, as will be shown in this section.

25.4.3.1 Genetic Encoding for Sensor Deployment

Since a candidate solution to the SDP requires a two-dimensional sensor ID matrix, a two-dimensional numeric encoding scheme needs to be adopted to make up the chromosomes instead of the conventional linear sequence in order. As Figure 25.6 shows, a sensor ID matrix for a possible sensor deployment scheme is constructed. Each element in the matrix on the right-hand side corresponds to a cell within a surveillance region on the left-hand side. As mentioned above, an empty value ε in the matrix indicates that its corresponding cell has no sensor deployed and should be covered by the sensors deployed in its neighborhood area. Furthermore, q types of available sensor are arranged in the following order:

$$\frac{d_{S_1}}{w(s_1)} \geq \frac{d_{S_2}}{w(s_2)} \geq \cdots \geq \frac{d_{S_k}}{w(s_k)} \geq \cdots \geq \frac{d_{S_q}}{w(s_q)}$$

© 2005 by Chapman & Hall/CRC

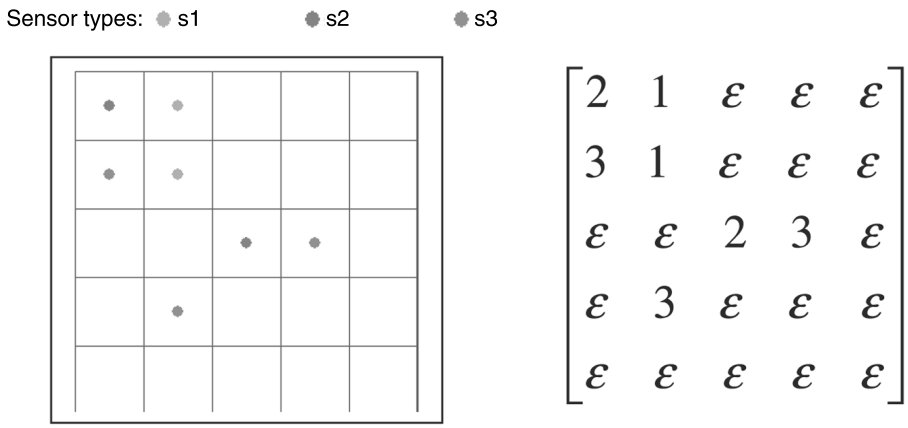


Figure 25.6. A candidate deployment solution and its corresponding matrix.

Recall that d_{S_k} and $w(s_k)$ are the maximum detection distance and cost of sensor of type k respectively. The rank of ratio is used to decide the probability of type of sensor selected during the population initialization, as the well as in the addition operation.

25.4.3.2 Fitness Function

The fitness function from the objective function can be constructed as

$$f(\Re) = P\{\Re | T \in R\} + g$$

where g, the penalty function for overrunning the constraint, is defined by

$$g = \begin{cases} 0 & \text{Cost}(\Re) \leq Q \\ \delta E_{\text{m}}(Q - \text{Cost}(\Re))/Q & \text{Cost}(\Re) > Q \end{cases}$$

such that δ is a proper penalty coefficient and is set to 100, and $E_{\text{m}} = \max_k \{d_{S_k}/w(s_k)\}$.

25.4.3.3 Selection of Candidates

The selection operation retains good candidates and eliminates others from the population based on the individual fitness values. It is also called a reproduction operation. It aims to inherit good individuals, either directly from the last generation or indirectly from the new individuals produced by mating the old individuals. The frequently used selection mechanisms include the fitness proportional model, the rank-based model, the expected value model, and the elitist model [22].

In this implementation, the survival probability B_i for each individual (solution) $\Re_i$ is computed based on the following fitness proportional model:

$$B_i = \frac{f(\Re_i)}{\sum_{j=1}^{M} f(\Re_j)}$$

where M is the population size. The hybridization individuals are produced according to the selection rule such that the individual with bigger B_i has a higher probability to survive.

© 2005 by Chapman & Hall/CRC

25.4.3.4 Implementation of Genetic Operators

The solution set of each new generation after the initial population is generated as follows. Randomly select two hybridization individuals $\Re_u$ and $\Re_v$ and combine them to get two other individuals $\Re'_u$ and $\Re'_v$ of the new generation by using combinatorial rules of crossover, mutation, inversion, translocation, addition, and deletion [23]. Some of these genetic operators are carried out on a two-dimensional basis. Except for crossover, all the other operators operate on only one parent solution. This process continues until all M individual solutions of the new generation are created.

1. *Crossover*. This is an operation of segment exchange for two solutions. Given two parent (hybridization individuals) solutions on the left side in Figure 25.7, a two-dimensional two-point crossover operation produces two child solutions on the right side, as Figure 25.7 illustrates. Both window size and location for crossover are selected randomly.
2. *Translocation*. The objective of translocation is to exchange information between different segments within a single solution, as Figure 25.8 illustrates. As in the crossover operation, the translocation window size is picked up at random, as well as its source and destination position.
3. *Mutation*. The mutation operator chooses one or more cells randomly in the surveillance region and changes their values by the preset mutation probability. Actually, it is a combination of addition and deletion operators. As Figure 25.9 shows, a sensor of type A is moved to a new place, a sensor of type B is replaced by a sensor of type C, a sensor of type C is deleted, and a sensor of type D is replaced by a sensor of type B. The selection probability of a certain type of sensor for the addition operator depends on the ratio of its detection range to unit price, so does for the population initialization.

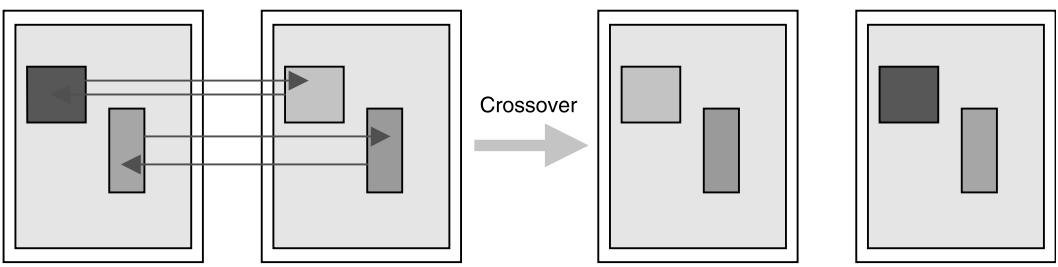


Figure 25.7. Two-dimensional two-point crossover.

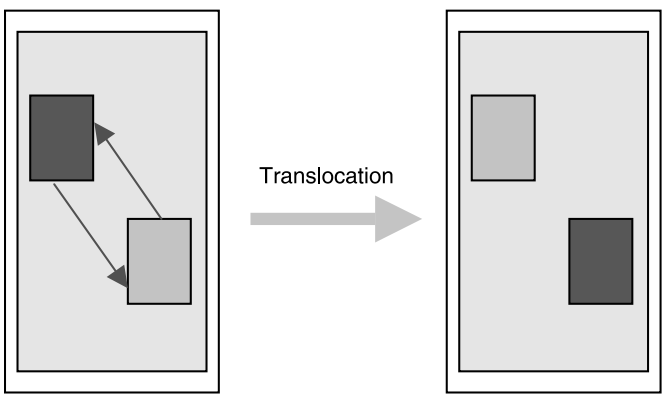


Figure 25.8. Two-dimensional translocation.

© 2005 by Chapman & Hall/CRC

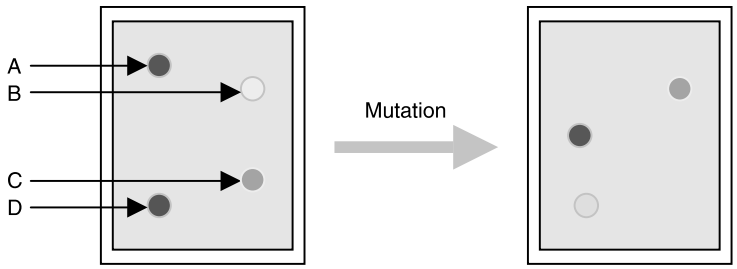


Figure 25.9. Mutation operator.

25.4.4 Computational Results

In this section, experimental test results comparing our genetic algorithm described in the last section with the greedy method based on uniform placement (UP) of sensors are presented. Both algorithms are implemented in C++. Consider that the target has a uniform *a priori* distribution in the surveillance region such that the probability of target T appearing in a cell $C(i,j)$ is $P\{T \in C(i,j)\} = 1/(mn)$. Then, the detection probability, which is the objective function, has the following formula by Equation (25.2):

$$P\{\Re | T \in R\} = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} P\{\Re | T \in C(i,j)\}/(mn)$$

with the constraint $\text{Cost}(\Re) \leq Q$. This formula is utilized for the computations in this section, and the case in which the target has other prior distributions can be handled using Equation (25.2) in place of the above expression.

In Case 1, consider a surveillance region of 50×50 cells with five types of sensors as listed in Table 25.1 All parameters used by the genetic algorithm are specified in Table 25.2 The investment limit is set to be 1800 units expense and the maximum generation number is set to be 200, also as shown in the upper data part of Figure 25.10 After 200 generations of optimization, an acceptable deployment scheme is achieved with detection probability of 94.52% for the surveillance region within the investment budget. The graph on the right side of Figure 25.10 shows the optimization process curve with the generation number represented on the x-axis and the corresponding fitness value represented on the y-axis. The computational result is illustrated on the left side of Figure 25.10 for the sensor deployment based on the genetic algorithm. In each cell, a local detection probability is given for evaluation.[3] Its corresponding three-dimensional display of the local coverage probabilities of 50×50 cells is shown in Figure 25.11.

Figure 25.12 shows the computational result of the same surveillance region based on UP using the sensors of type with the maximum ratio of detection range to unit price. The UP achieves an average detection probability 88.83% within the investment budget. Its corresponding three-dimensional display of the local coverage probabilities of 50×50 cells is shown in Figure 25.13.

Figure 25.14 is obtained by carrying out a series of runs for this surveillance region with increasing investment limits. From the plot of detection probability versus investment limit shown in Figure 25.14, it is observed that increasing the investment beyond 1800 units does not pay off any more, since the incremental gain of the detection probability is very small. Not only could this graph help in determining a sensor deployment scheme within any given cost for a given surveillance region, but it could also be used to choose a proper initial investment limit for the given surveillance region.

[3]The values of local detection probabilities are overwritten by those values of neighbor cells except for the rightmost column due to the relatively small display screen, as is the case in Figure 25.11.

© 2005 by Chapman & Hall/CRC

Table 25.1. Parameter specifications for five types of sensors used in Case 1

Sensor type	Sensor ID	Unit price	Detection range	Detection coefficient
Sen1	1	86	124	80
Sen2	2	111	159	78
Sen3	3	113	163	68
Sen4	4	135	195	68
Sen5	5	139	200	84

Table 25.2. Parameters used by the genetic algorithm in Case 1

Genetic algorithm parameters	Values
Maximum generation number	200
Maximum investment limit	1800
Population size	30
Probability of crossover	0.99
Probability of mutation	0.24
Probability of deletion	0.10
Probability of translocation	0.99
Probability of inversion	0.82
Probability of addition	0.10

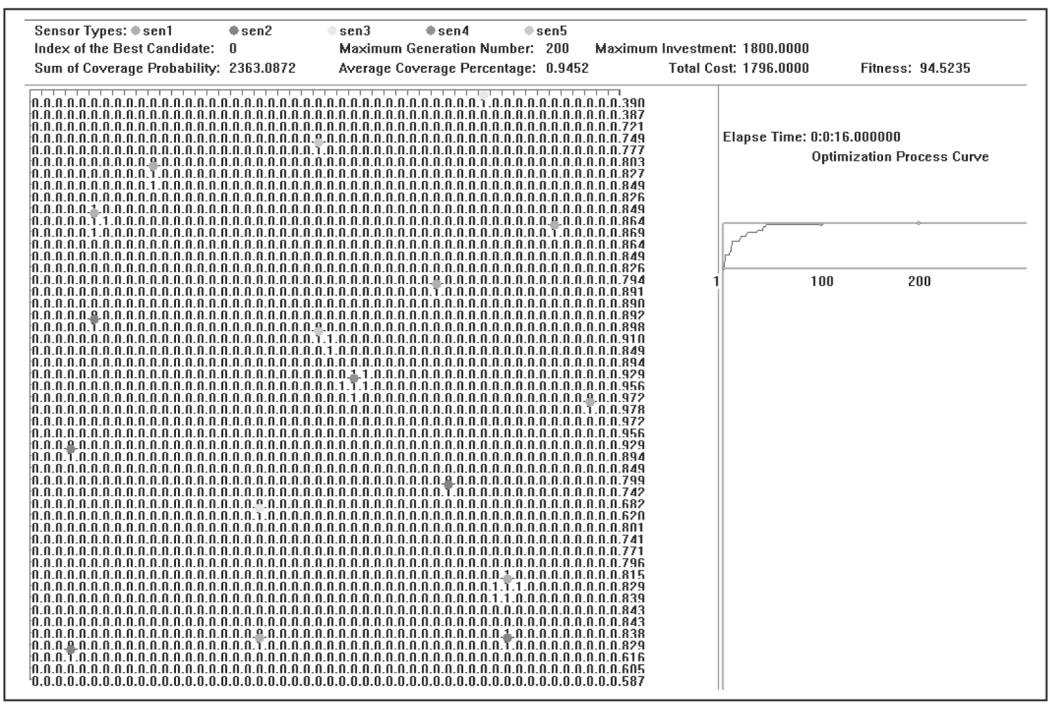


Figure 25.10. Test result for a surveillance region with 50×50 cells based on the genetic algorithm.

Now let us consider larger surveillance regions and more sensor types with different parameters. The UP always uses only the sensors of type with the maximum ratio of detection range to unit price. The comparisons of computational results between the genetic algorithm and the UP are shown in Table 25.3. In summary, the genetic algorithm achieved higher probability of detection while satisfying the cost bound.

© 2005 by Chapman & Hall/CRC

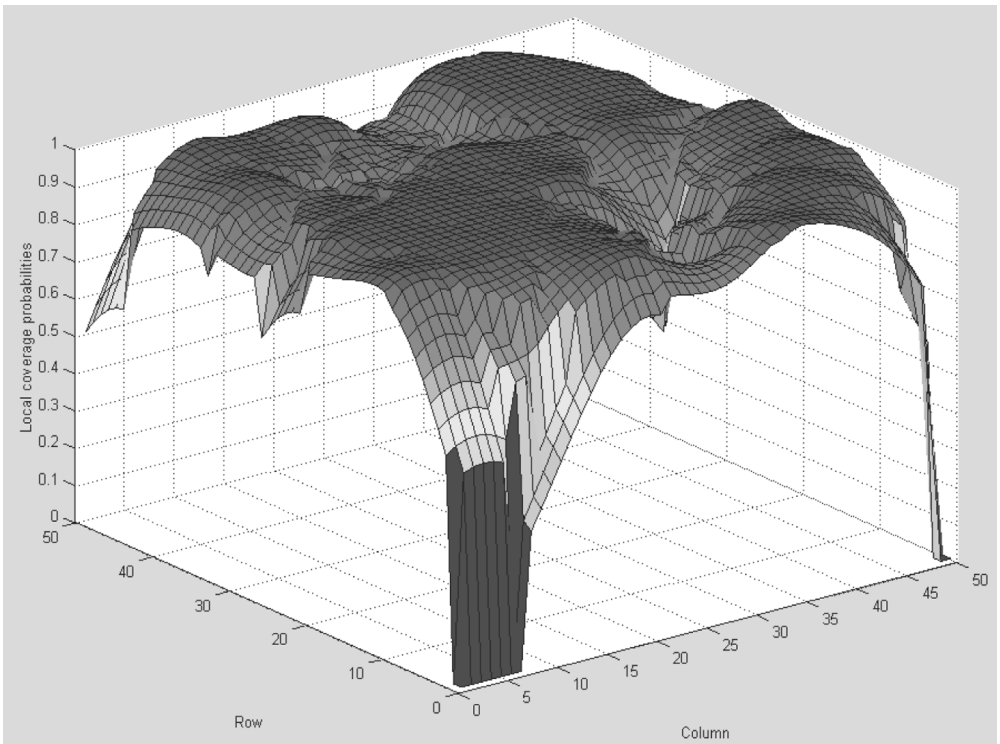


Figure 25.11. Three-dimensional display of the local coverage probabilities of 50×50 cells computed by the genetic algorithm.

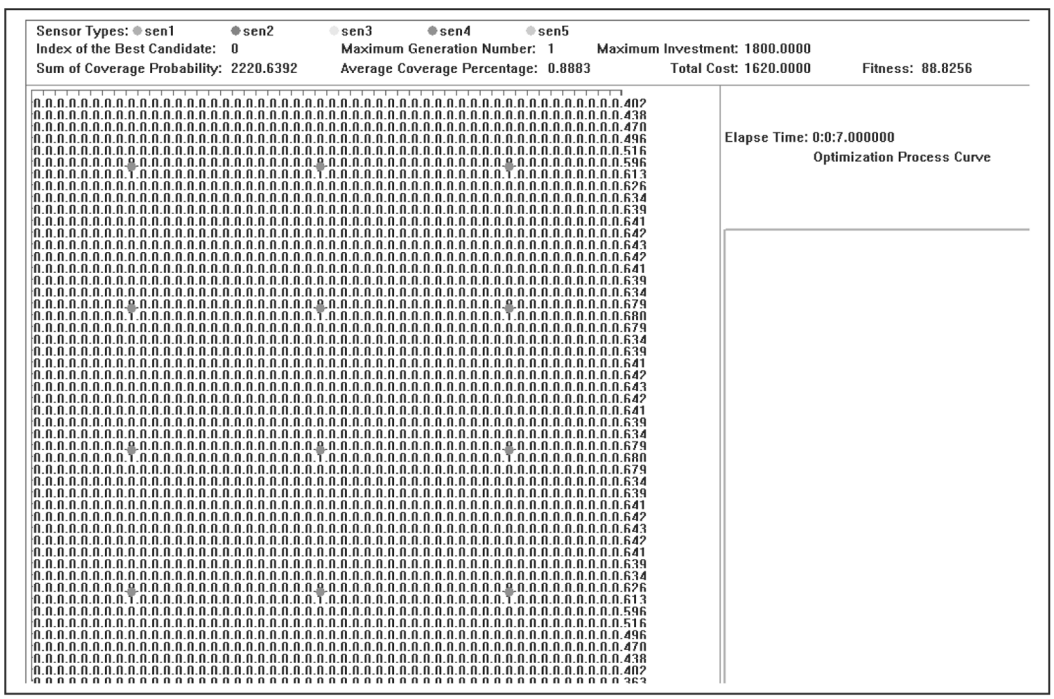


Figure 25.12. Test result for a surveillance region with 50×50 cells based on UP.

© 2005 by Chapman & Hall/CRC

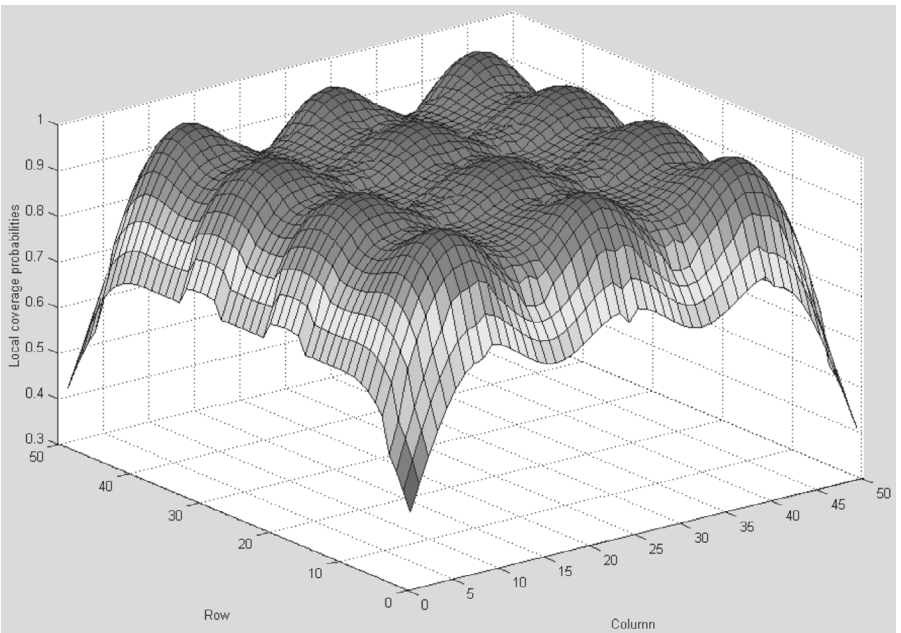


Figure 25.13. Three-dimensional display of the local coverage probabilities of 50×50 cells computed by UP.

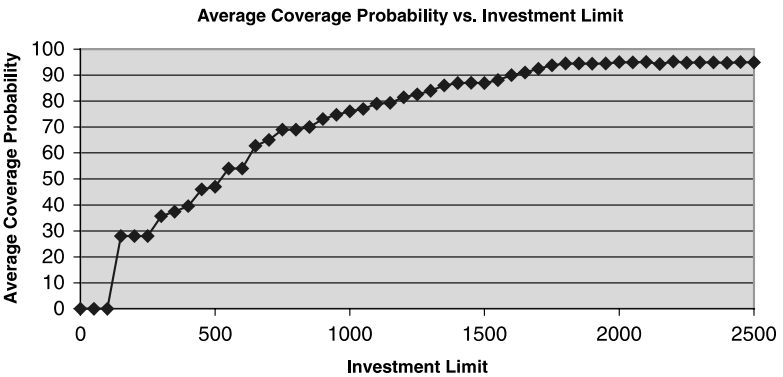


Figure 25.14. Detection probability versus investment limit for a region with 50×50 cells.

Table 25.3. Comparison of performance of genetic algorithm and UP

Surveillance region size	Number of sensor types	Maximum investment limit	Genetic algorithm		Uniform placement	
			Total cost	Ave. detection probability (%)	Total cost	Ave. detection probability (%)
50×50	5	1800	1796	94.52	1620	88.83
100×100	5	2100	2081	93.03	1920	84.41
120×120	7	2250	2226	94.18	2160	87.44
150×150	7	2350	2340	93.12	2187	85.96
200×200	8	2600	2587	93.97	2430	88.61
300×300	5	3900	3861	93.64	3630	88.75
600×600	6	4600	4598	96.84	4400	87.69
750×750	5	6000	5995	93.81	5670	87.40
900×900	8	9000	8949	95.16	8993	86.61
1000×1000	9	9700	9698	93.70	9680	88.58

© 2005 by Chapman & Hall/CRC

25.4.5 Conclusions

Optimal detection and target localization are two critical but difficult tasks of sensor deployment, particularly if the sensors are of different types and incur different costs. A general SDP for a planar grid region is formulated with the objective of maximizing the detection probability within a given deployment cost. It is shown that this problem is NP-complete, and then an approximate solution is presented using a genetic algorithm for the case where the sensor distributions are statistically independent. Computational results are presented when the target has uniform prior distribution and Gaussian approximation for sensor distributions, which shows that this solution performs favorably in solving the SDP. This solution is applicable to more general cases in which the target's *a priori* distribution is not uniform and the sensor distributions are more complicated but easily computable. In general, however, the computational cost of such extensions would be correspondingly higher.

There are a number of avenues for further research. First, it would be interesting to see whether analytical performance bounds can be placed on the solution computed by our method. Also, extensions of the proposed method when the statistical independence is not satisfied would be applicable to wider classes of sensor deployment problems. The challenge in this case is to ensure low computational complexity by utilizing the domain-specific knowledge of the sensors. In particular, the simple incremental formula in Equation (25.1) is no longer valid, and in the worst case this computation may have an exponential complexity for arbitrary distributions. From an algorithmic perspective, polynomial-time (deterministic) approximations to the sensor deployment problem that are guaranteed to be provably close-to-optimal will be of future interest.

Acknowledgment

This research is sponsored by the Material Science and Engineering Division, Office of Basic Energy Sciences, U.S. Department of Energy, under contract No. DE-ACC5-00OR22725 with UT-Battelle, LLC, the Defence Advanced Projects Research Agency under MIPR No. K153, and by National Science Foundation under Grants No. ANI-0229969 and No. ANI-335185.

References

[1] Dhillon, S.S. and Chakrabarty, K., A fault-tolerant approach to sensor deployment in distributed sensor networks, in *The 23rd Army Science Conference*, Orlando, FL, December, 2–5, 2002.

[2] Chakrabarty, K. *et al.*, Coding theory framework for target location in distributed sensor networks, in *Proceedings of International Symposium on Information Technology: Coding and Computing*, 2001, 130.

[3] Convay, J.H. and Guy, R.K., *The Book of Numbers*, Copernicus Books, New York, 1996, 220.

[4] Chakrabarty, K. *et al.*, Grid coverage for surveillance and target location in distributed sensor networks, *IEEE Transactions on Computers*, 51(12), 1448, 2002.

[5] Chakrabarty, K. *et al.*, Coding theory framework for target location in distributed sensor networks, in *Proceedings of the IEEE International Conference on Information Technology*: *Coding and Computing*, Las Vegas, NV, April 2001, 157.

[6] Bulusu, N. *et al.*, Adaptive beacon placement, in *Proceedings of ICDCS-21*, Phoenix, AZ, USA, April 2001.

[7] Bulusu, N. and Estrin, D., Scalable *ad hoc* deployable RF-based localization, in *Proceedings of the Grace Hopper Celebration of Women in Computing Conference 2002*, Vancouver, British Columbia, Canada, October 2002.

[8] Guibas, L. *et al.*, Visibility-based pursuit evasion in a polygonal environment, *International Journal of Computational Geometry Applications*, 9(4/5), 471, 1999.

[9] Meguerdichian, S. *et al.*, Coverage problems in wireless *ad hoc* sensor networks, in *Proceedings of IEEE Infocom 2001*, Anchorage, AK, April 22–26, 2001.

© 2005 by Chapman & Hall/CRC

[10] Iyengar, S.S. *et al.*, *Advances in Distributed Sensor Integration*: *Application and Theory*, Prentice Hall, New Jersey, 1995, 130.
[11] Brooks, R.R. and Iyengar, S.S., *Multi-Sensor Fusion: Fundamentals and Applications with software*, Prentice Hall PTR, Upper Saddle River, NJ, 1998.
[12] Liu, Y. *et al.*, Non-numerical parallel algorithms (II), *Genetic Algorithms*, Science Press, Beijing, 1995.
[13] Han, Z.X. and Wen, F.S., Optimization method simulating evolution and its application, *Computer Science*, 22(2), 1995.
[14] A compendium of NP optimization problems, http://www.nada.kth.se/~viggo/problemlist/compendium.html.
[15] http://www.nosc.mil/robots/research/manyrobo/detsensors.html.
[16] Garey, M.R. and Johnson, D.S., *Computers and Intractability*: *A Guide to the Theory of NP-Completeness*, W.H. Freeman, 1979.
[17] http://www.nlectc.org/perimetr/full2.htm.
[18] Liu, C.L., *Introduction to Combinatorial Mathematics*, McGraw-Hill, 1968.
[19] Holland, J.H. *Adaptation in Nature and Artificial Systems*, University of Michigan Press, 1975 (Reprinted by MIT Press 1992).
[20] Coley, D.A., *An Introduction to Genetic Algorithms for Scientists and Engineers*, World Scientific, 1999.
[21] Winston, P.H., *Artificial Intelligence*, 3rd ed., Addison-Wesley, 1993.
[22] Chen, G.L., *Genetic Algorithm and its Application*, People's Post Publishing House, China, 1996.
[23] Goldberg, D.E., *Genetic Algorithms in Search*, *Optimization & Machine Learning*, Addison-Wesley, 1989.

© 2005 by Chapman & Hall/CRC

26

Genetic Algorithm for Mobile Agent Routing in Distributed Sensor Networks*

Qishi Wu, S.S. Iyengar, and Nageswara S.V. Rao

26.1 Introduction

In the past decade, sensor networks have become an active area of research for computer scientists and network engineers owing to their wide usage in both military and civilian applications [1]. The increasing sophistication of multi-sensor systems for state estimation, region surveillance, and target detection and tracking has generated a great deal of interest in the development of new computational structures and networking paradigms [2].

A distributed sensor network (DSN) consists of intelligent sensors that are geographically dispersed in a region of interest and interconnected via a communication network. The sensed data of different types (such as acoustic, seismic, and infrared, etc.) are preprocessed by sensor nodes and then transmitted over the network to participate in the data integration at processing elements, based on which appropriate inferences can be derived about the environment for certain purposes. The study of information fusion methods has been the research focus since the early stage of DSN development [3–5]. Recent advances in sensor technologies make it possible to deploy a large number of inexpensive and small sensors to "achieve quality through quantity" in very complex scenarios, which necessitates applying new computing techniques such as genetic algorithms (GAs) to meet some theoretical and methodological challenges. This chapter provides a general introduction to GAs and their application to the mobile agent routing problem in a DSN with a special networking paradigm.

DSNs are typically deployed for remote operations in large unstructured geographical areas, where wireless networks with low bandwidth are usually the only means of communication among the sensors.

*A dynamic version of this problem with additional results is presented by Wu, Q., Rao, N.S.V., Barhen, J., Iyengar, S.S., Vaishnavi, V.K., Qi, H. and Chakrabarthy, K., "On computing mobile agent routes for data fusion in distributed sensor networks," *IEEE Transactions* on *Knowledge* and *Data Engineering*, 16(6), 740, 2004.

© 2005 by Chapman & Hall/CRC

The communication consumes the limited power available at sensor nodes, and thus power consumption is to be restricted. Furthermore, the massively deployed sensors usually supply huge amounts of data of various modalities, which makes it critical to collect only the information that is most desired and to collect it efficiently. Despite the abundance of sensors deployed, not all the information from these sensors needs to be collected to ensure the quality of the fused information, such as adequate detection energy for target detection or tracking.

Instead of sending all sensor data to the processing element, which performs a one-time data fusion as in a conventional server/client system, the mobile agent-based DSN (MADSN) proposed by Qi *et al.* [6] enables the computation to be spread out onto the participating leaf nodes with the aim of decreasing the consumption of scarce network resources (mostly the bandwidth) and the risk of being spied upon with hostile intent. In such a network scheme, a mobile agent carrying the executable instructions of data integration is dispatched from the processing element and selectively visits the leaf sensors along a certain path to fuse the data incrementally on a sequential basis.

The path quality of a mobile agent significantly affects the overall performance of MADSN implementation because the communication cost and detection accuracy depend on the order and the number of nodes to be visited. We formulate the mobile agent routing problem (MARP) as a combinatorial optimization problem with certain constraints and construct an appropriate objective function that reflects the routing requirements. We show the MARP's NP-hardness by reducing to it a variation of the three-dimensional traveling salesman problem, which rules out any polynomial solutions. Therefore, we propose an approximate solution based on a two-level GA and compare the simulation results with those computed by two other deterministic heuristics, namely local closest first (LCF) and global closest first (GCF).

The rest of this chapter is organized as follows. In Section 26.2, we introduce a general computing technique based on GAs. In Section 26.3, we describe the models for sensor nodes and wireless communication links and then formulate the MARP. The details of the solution using a GA are given in Section 26.4, including the design of a two-level encoding scheme, derivation of the objective function, and implementations of genetic operators. Simulation results are presented and discussed in Section 26.5. Concluding remarks are provided in Section 26.6.

26.2 Computational Technique Based on GAs

26.2.1 Introduction to GAs

A GA is a computational model simulating the process of genetic selection and natural elimination in biologic evolution. Pioneering work in this field was conducted by Holland in the 1960's [7,8]. GAs were proposed to find global or local optima in a large search space. Compared with traditional search algorithms in artificial intelligence, a GA is automatically able to acquire and accumulate the necessary knowledge about the search space during its search process, and self-adaptively control the entire search process through random optimization techniques. A computational technique based on a GA is particularly useful to avoid combinatorial explosion, which is always caused by disregarding the inherent knowledge within the enormous search space. In addition, the GA is characterized by its simplicity, flexibility, robustness, and adaptability to parallel processes. It has found many successful applications in various areas solving combinatorial optimization problems and nonlinear problems with complicated constraints or nondifferentiable objective functions.

26.2.2 A General Method Using GAs

Computation with a GA is an iterative process that simulates the process of genetic selection and natural elimination in biologic evolution. In each iteration cycle, good candidate solutions are retained and any unqualified solutions are screened out according to their corresponding fitness values. Genetic operators, such as crossover, mutation, translocation, and inversion, are then performed on those

© 2005 by Chapman & Hall/CRC

surviving solutions to produce a next generation of new candidate solutions. The above process is carried out repeatedly until a certain convergence condition is met. To illustrate the principle of the algorithm, we take the classical knapsack problem as an example [9], which is formulated as follows:

$$\text{Maximize: } \sum_{i=1}^{n} B_i X_i \tag{26.1}$$

$$\text{Constraint: } \sum_{i=1}^{n} S_i X_i \le C \quad X_i \in \{0,1\},\ 1 \le i \le n \tag{26.2}$$

where S_i represents the resource consumption for the ith activity, C represents the total available resources, and B_i represents the profit gained from the ith activity. X_i holds binary values: if the ith activity is carried out, $X_i = 1$; otherwise, $X_i = 0$. The essence of the knapsack problem is to pursue the maximum profit with the constraint of limited total available resources. We now describe the standard steps taken by a GA to find a solution to the knapsack problem.

1. Initialization: a set of M random solutions $T_k (1 \le k \le M)$ are generated, where M is an appropriately selected initial population size.
2. Genetic encoding: a string T of n binary bits is used to represent one possible solution. If the ith activity is carried out, $T(i)(1 \le i \le n) = 1$; otherwise $T(i)(1 \le i \le n) = 0$.
3. Fitness value calculation: the objective function of the knapsack problem can be defined as

$$J(T_k) = \sum_{i=1}^{n} T_k(i) B_i \tag{26.3}$$

$$\text{subject to: } \sum_{i=1}^{n} T_k(i) S_i \le C \quad 1 \le k \le M \tag{26.4}$$

   where C is a bounding constant. We construct a fitness function for the knapsack problem as follows to compute the fitness value for each individual solution:

$$f(T_k) = J(T_k) + g(T_k) \quad 1 \le k \le M \tag{26.5}$$

   where $g(T_k)$ is the penalty function when T_k violates the constraints, which may take the following form:

$$g(T_k) = \begin{cases} 0 & C \ge \sum_{i=1}^{n} T_k(i) S_i \\ \beta E_{\mathrm{m}} \left(C - \sum_{i=1}^{n} T_k(i) S_i \right) & C < \sum_{i=1}^{n} T_k(i) S_i \end{cases} \tag{26.6}$$

   where E_{m} is the maximum value of $B_i/S_i\,(1 \le i \le n)$, and β is a proper penalty coefficient.
4. Survival probability calculation: the survival probability P_k for each individual (solution) T_k can be calculated based on the following fitness proportional model:

$$P_k = f(T_k) / \sum_{j=1}^{M} f(T_j) \tag{26.7}$$

   A random selector is then designed to produce the hybridization individuals according to each P_k.

© 2005 by Chapman & Hall/CRC

5. New generation production: two hybridization individuals T_u and T_v are combined to create two individuals T'_u and T'_v of new generation by applying combinatorial rules of selection, crossover, mutation, and inversion. This process continues until all M individual solutions of the new generation are produced. There are several genetic operators involved in this procedure.
   Crossover is an operation of segment exchange for two solutions. Given two parents (hybridization individuals) solutions with their crossover points represented by "/":

   $$T_u = 01010/1011100$$
   $$T_v = 10100/1101010$$

   a one-point crossover operator produces two children solutions:

   $$T'_u = 01010/1101010$$
   $$T'_v = 10100/1011100$$

   Inversion is to reverse the order of data in a solution segment. Given one parent solution with the inversion segment enclosed by a pair of "/":

   $$T_u = 010/01101/1100$$

   an inversion operator produces the following child:

   $$T'_u = 010/10110/1100$$

   The mutation operator chooses one or more gene loci randomly in the individual string and changes their values (i.e. 0–1 reverses) with the preset mutation probability. Given one parent solution as follows:

   $$T_v = 010101011100$$
   $$\quad\quad\uparrow \quad\quad \uparrow$$

   two gene loci before mutation

   a two-point mutation operator produces the following child:

   $$T'_v = 011101010100$$
   $$\quad\quad\uparrow \quad\quad \uparrow$$

   two gene loci after mutation

6. Repeat Steps (2) to (5) until the predefined convergence condition is met, e.g. the maximum generation number is reached or the solution quality is satisfied.

A general description of the above iterative process is given in C language as follows:

```
main()
  {
    int gen_no;
    initialize();
    generate(oldpop);
    for(gen_no=0;gen_no<maxgen;gen_no++)
```

© 2005 by Chapman & Hall/CRC

```
  {
    evaluate(oldpop);
    newpop=select(oldpop);
    crossover(newpop);
    mutation(newpop);
    inversion(newpop);
    oldpop=newpop;
  }
}
```

The above pseudo code only depicts the major steps of a GA. Some auxiliary functions are needed to implement a complete GA for a certain application. During the search process, the GA does not require any outside knowledge except the fitness values to select qualified solutions. Therefore, the design of the fitness function has a significant impact on the overall algorithmic performance.

26.2.3 Parameters and Operators in a GA

26.2.3.1 Population Size

Population size is a key parameter in a GA. The schema theorem [10] establishes that, given the population size M, the genetic operators are able to produce M^3 schemas, which ensures that the number of building blocks is dramatically increased when the search for the optimal solution progresses. Obviously, a GA with a larger population size is more likely to obtain the global optimum because a larger population size produces a wider variety of individuals and, therefore, the search process has a higher probability of avoiding being trapped into local optima. On the contrary, a small population size limits the search space, and hence premature convergence may occur under this circumstance, which may greatly impair the performance.

However, a large population size also brings some disadvantages. For instance, the computation complexity increases as a result, and some good individuals with high fitness values may be eliminated during the selection operation.

26.2.3.2 Crossover, Mutation, and Inversion

Similar to gene recombination, which plays an essential role during the natural biologic evolution process, crossover in a GA is the most critical operator in the genetic search strategy, which guides the main behavior of the optimization process. There are several commonly used crossover schemes, such as one-point crossover, two-point crossover, and multi-point crossover. A good design of any crossover operator must ensure that the desirable gene segments of old individuals be properly inherited by the new individuals of the new generation. A high crossover probability may improve the GA's capability to explore new solution space, while increasing the likelihood of disordering the combination of good gene segments. An inappropriately low crossover probability may cause the search process to be trapped in a dull status and be prone to ceasing.

The main purpose of a mutation operator is to maintain the variety of the population by preventing a single important gene segment from being corrupted. In practice, a relatively small mutation probability, such as 0.001, is favorable because the GA may tend to be a random search if too frequent a mutation operation is conducted.

Inversion is actually a special form of mutation. It is designed to carry out a reordering operation and improve the local search ability. Neither crossover nor mutation is adequate for searching in the local solution space: the search activities of the crossover operator span the whole feasible solution space, and the local search ability of the mutation operator is always suppressed by genetic selection and natural elimination.

© 2005 by Chapman & Hall/CRC

26.2.3.3 Encoding and Fitness Function

Since a GA is unable to manipulate the parameters in the problem space directly, it is necessary to convert them to individuals made up of genes in the GA domain. This mapping from the problem space to the algorithm domain is called encoding. Actually, the robustness of the GA reduces the reliance of performance on encoding schemes, as long as a minimum of three encoding criteria, such as completeness, soundness, and non-redundancy, are satisfied [11].

In general, the control of the search process in a GA needs no information from outside except the fitness values (or objective values). The objective function of a complex system usually has discontinuous or nondifferentiable constraints. For a general optimization problem with complicated constraints, the penalty method is often used in the design of a fitness function. For example, an original minimization problem with constraints can be described as follows [10]:

$$\text{Minimize: } F(x) \tag{26.8}$$

$$\text{with constraints: } b_l(x) \geq 0 \quad l = 1, 2, \ldots, p \tag{26.9}$$

where $F(x)$ is the objective function and $b_l(x)$ are a group of constraint functions. By applying the penalty method, we are able to convert the above problem to a nonconstraint problem:

$$\text{Minimize: } F(x) + \lambda \sum_{l=1}^{p} \Phi[b_l(x)] \tag{26.10}$$

where λ is the penalty coefficient and Φ is the penalty function, which may take the form of Equation (26.6).

26.2.3.4 Selection Mechanism

The selection operation, also referred to as reproduction operation, is to select good individuals and eliminate bad individuals from the population according to individual fitness values. A good selection mechanism is able to inherit good individuals directly from last generation or indirectly from the new individuals produced by mating the old individuals. The commonly used selection mechanisms include fitness proportional model, rank-based model, expected value model, and elitist model, etc.

26.3 The MARP

We now briefly describe the architecture of an MADSN to motivate the later formulation of the optimization problem. An MADSN typically consists of three types of component: *processing elements* (PEs), s*ensor nodes*, and *communication network* [12]. The various processing elements and sensors are usually interconnected via a wireless communication network. A group of neighboring sensor nodes that are commanded by the same PE forms a *cluster*.

26.3.1 Sensor Nodes

A sensor node, also referred to as a leaf node, is the basic functional unit for data collection in an MADSN. A sensor node may have several channels with different sensors connected to each of them. Sensor nodes are always geographically distributed to collect different types of measurement, such as acoustic, seismic, and infrared, from the environment. The data acquisition is controlled by a sampling subsystem, which provides the acquired data to the main system processor [13]. The signal energy from each channel can be detected individually and processed in the analog front end. A mobile agent migrates among the sensor nodes via the network, integrates local data with a desired resolution

© 2005 by Chapman & Hall/CRC

sequentially, and carries the final result to the originating PE. The fused data may be used to derive appropriate inferences about the environment for a certain civilian or military application.

We now provide object-oriented descriptions of sensor and PE nodes, which are used in our implementation. The sensor label is a unique ID of a sensor node, which corresponds to its static Internet protocol (IP) address in the sensor network. We assume that a PE with label "0" remains active during the period of operation of the MADSN. Some sensor nodes may be shut down or go to sleep due to intermittent faults or power considerations, and may be brought back up later if necessary. The sensor location is determined by its longitude and latitude obtained from the embedded global position system (GPS) module. The abstract sensor class, defined in the C++ language, is listed in Appendix A. Both the leaf node and the PE are derived from the abstract sensor class.

The signal energy, which is detected in real time at each local node and broadcast over the whole cluster, is an indicator of how close the node is to a potential target. In target detection and tracking applications, a leaf node with higher signal energy carries more information and should have higher priority of being visited. To simplify computation, we use a quantitative value to represent the level of signal energy detected by a local sensor node. The *setup* time spent by a PE accounts for loading the mobile agent code and performing other initialization tasks.

26.3.2 Communication Links

Wireless communication links need to be established between neighboring nodes as the mobile agent migrates along a route. The embedded RF modems of a sensor node provide such a networking capability with low power requirement. On the WINS NG 2.0 platform, each node is equipped with two RF modems, both of which support 2.4 GHz frequency-hopped spread spectrum (FHSS) communication [13]. The different clusters select different "network numbers," which correspond to separate hopping pseudo-noise sequences to avoid interferences. The detailed radio configuration and wireless link establishment is beyond the scope of this chapter. We define an abstract link class with only the parameters we are interested in (see Appendix A for details).

It is worthwhile noting that the message transmission time between two sensor nodes depends not only on the physical distance between them, but also on the channel bandwidth, the data packet loss rate, and the size of messages to be transmitted, which includes partially integrated data and mobile agent code itself. In general, the electromagnetic propagation time is almost negligible in short-range wireless communication. Hence, the physical distance is not explicitly considered in our model, but it is incorporated as a part of the path loss representing the signal attenuation. The received signal strength below a certain level due to path loss may not be acceptable. The system loss factor is a parameter of the free-space propagation model, which is not necessarily related to the physical propagation [14].

26.3.3 Mobile Agent Routing

A mobile agent is dispatched from the PE and is expected to visit a subset of sensors within the cluster to fuse data collected in the coverage area. Generally speaking, the more sensors visited, the higher will be the accuracy achieved using any reasonable data fusion algorithm [15]. It is important to select an appropriate route so that the required signal energy level can be achieved with a low cost in terms of total energy consumption and path loss.

A MADSN with a simple configuration is shown in Figure 26.1 for illustrative purposes. The sensor network contains one PE, labeled as S_0, and $N = 10$ leaf nodes, labeled as $S_i, i = 1, 2, \ldots, N$, one of which is down. The sensor nodes are spatially distributed in a surveillance region of interest, each of which is responsible for collecting measurements in the environment. The signal energy detected by sensor node S_i is denoted by e_i, $i = 1, 2, \ldots, N$. Sensor node S_i takes time $t_{i,\text{acq}}$ for data acquisition and time $t_{i,\text{proc}}$ for data processing. The wireless communication link with physical distance $d_{i,j}$ between sensor node S_i and S_j has channel width W bits and operates at frequency B Hz. Some sensor nodes may be down temporarily due to intermittent failures, as sensor S_9 in Figure 26.1.

© 2005 by Chapman & Hall/CRC

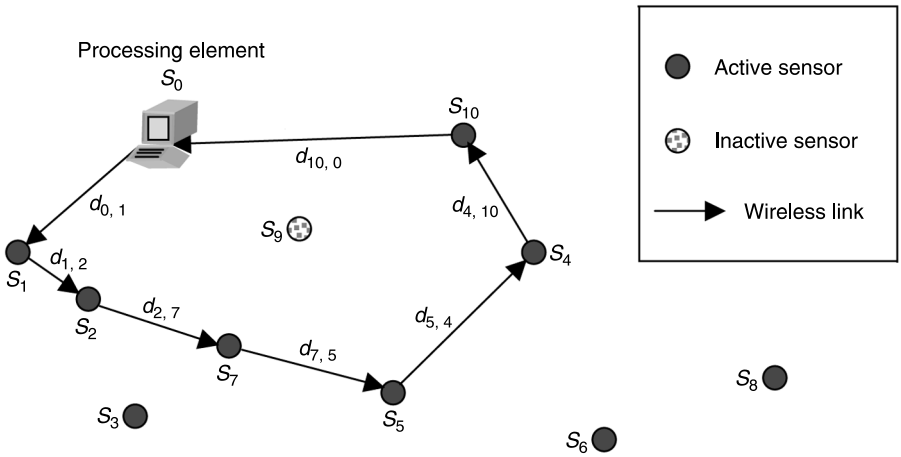


Figure 26.1. A simple MADSN with one PE and ten leaf nodes.

The routing objective is to find a path for a mobile agent that satisfies the desired detection accuracy, while minimizing the energy consumption and path loss. The energy consumption depends on the processor operational power and computation time, and the path loss is directly related to the physical length of the path selected. We define these quantities next.

26.3.4 Objective Function

The objective function for the MARP is based on three aspects of a routing path: energy consumption, path loss, and detected signal energy.

1. *Energy consumption*. The energy consumption at a sensor node is determined by the processing speed and the computation time. If an energy-driven-real-time operating system (RTOS) is installed on the sensor node, then the processor speed can be scaled dynamically depending on workload and task deadlines [16]. For wireless message transmissions, the energy consumption depends on the sensor's transmission power and message transmission time. We assume that the message includes the mobile agent code of size M bits and measured data of size D bits. For a given desired resolution, a fixed data size D is used to store the partially integrated data at each sensor. The time for the message to be transmitted over a wireless channel of bandwidth BW bps is calculated as

$$t_{\text{msg}} = \frac{1}{B}\left\lceil \frac{M+D}{W} \right\rceil \tag{26.11}$$

The energy consumption EC of path P, consisting of nodes $P[0], P[1], \ldots, P[\text{H}-1]$, is defined as

$$\begin{aligned} \text{EC}(P) = {} & a(t_{0,\text{setup}} + t_{0,\text{proc}})F_0^2 + P_{0,t}\, t_{\text{msg}} \\ & + \sum_{k=1}^{H-1} \left\{ b\left[(t_{P[k],\text{acq}} + t_{P[k],\text{proc}})F_{P[k]}^2 \right] + P_{P[k],t}\, t_{\text{msg}} \right\} \end{aligned} \tag{26.12}$$

where kth leaf node $S_{P[k]}$ on path P has data acquisition time $t_{P[k],\text{acq}}$, data processing time $t_{P[k],\text{proc}}$, operational level $F_{P[k]}$, transmitting power $P_{P[k],t}$, $i = 1, 2, \ldots, N$, and node $P[0] = 0$ corresponds to the PE. Coefficients a and b are chosen to "normalize" the processor speed to its power level.

© 2005 by Chapman & Hall/CRC

2. *Path loss*. The power received by sensor S_j has the following relation with the power transmitted by sensor S_i according to the Friis free-space propagation model [14]

$$P_{j,\mathrm{r}}(d_{i,j}) = P_{i,\mathrm{t}} \frac{G_{i,\mathrm{t}} G_{j,\mathrm{r}} \lambda^2}{(4\pi)^2 d_{i,j}^2 \beta} \tag{26.13}$$

where $G_{i,\mathrm{t}}$ is the gain of sensor S_i as a transmitter and $G_{j,\mathrm{r}}$ is the gain of sensor S_j as a receiver. Wavelength λ is the ratio of speed of light c and carrier frequency f, and β is the system loss factor. The physical distance $d_{i,j}$ between S_i and S_j is computed from their spatial locations. Path loss (PL) represents the signal attenuation as a positive quantity measured in decibels, and is defined as the difference (in decibels) between the effective transmitted power and the received power:

$$\mathrm{PL}(d_{i,j}) = 10 \log \frac{P_{i,\mathrm{t}}}{P_{j,\mathrm{r}}} = 10 \log \left[\frac{(4\pi)^2 \beta}{G_{i,\mathrm{t}} G_{j,\mathrm{r}} \lambda^2} d_{i,j}^2 \right] \tag{26.14}$$

Therefore, the total path loss along path P can be calculated as

$$\mathrm{PL}(P) = \sum_{k=0}^{H-1} \left[10 \log \left(\frac{(4\pi)^2 \beta}{G_{P[k],\mathrm{t}} G_{P[(k+1) \bmod H],\mathrm{r}} \lambda^2} d_{P[k],P[(k+1) \bmod H]}^2 \right) \right] \tag{26.15}$$

3. *Signal energy*. An active sensor detects a certain amount of energy emitted by the potential target, which may or may not be used by a mobile agent for data integration. A mobile agent always tries to accumulate as much signal energy as possible for a accurate decision in target classification or tracking application. The sum of the detected signal energy SE along path P is defined as:

$$\mathrm{SE}(P) = \sum_{k=1}^{H-1} s_{P[k]} \tag{26.16}$$

where $s_{P[k]}$ is the signal energy detected by the *k-th* sensor node on path P.

By combining the above three aspects of a routing path, we consider an objective function as follows:

$$O(P) = \mathrm{SE}(P) \left(\frac{1}{\mathrm{EC}(P)} + \frac{1}{\mathrm{PL}(P)} \right) \tag{26.17}$$

wherein three terms $\mathrm{SE}(P)$, $\mathrm{EC}(P)$, and $\mathrm{PL}(P)$ are first normalized to reflect appropriately the contributions by various loss terms. This objective function prefers paths with higher signal energies by penalizing those with high path losses and energy consumption. A path providing high signal energy at the expense of a considerable amount of energy consumption and path loss may not be preferable. Alternative objective functions may be used as long as they correctly reflect the tradeoff between detected signal energy, energy consumption, and path loss.

To facilitate the GA in Section 26.4, we define a fitness function based on the objective function as follows:

$$f(P) = O(P) + g \tag{26.18}$$

© 2005 by Chapman & Hall/CRC

where g is the penalty function for overrunning the constraint, defined by

$$g = \begin{cases} 0 & \mathrm{SE}(P) \geq E \\ \delta(\mathrm{SE}(P) - E)/E & \mathrm{SE}(P) < E \end{cases} \tag{26.19}$$

where E is the desired detection accuracy or signal energy level and δ is a properly selected penalty coefficient.

26.3.5 NP-Hardness of MARP

The MARP is to compute a path P in an MADSN such that $O(P) > \tau$ and the k-hop MARP (k-MARP) additionally requires that the path P have exactly k edges. We now show the latter to be NP-hard by reducing the three-dimensional maximum traveling salesman problem (MTSP) to it, which is an indication of the intractability of the MARP. We first present the definition of the MTSP. Given a completely connected graph $G = (V, E)$, and a nonnegative real number α, does there exist a closed-loop path P, with nodes $P[0], P[1], \ldots, P[n-1], P[n] = P[0]$, such that $\sum_{i=0}^{n-1} l_3(P[i], P[(i+1) \bmod n]) \geq \alpha$? Here, each vertex corresponds to a point in three-dimensional Euclidean space $\mathbb{R}^3$. The starting point $v_{P[0]}$ and ending point $v_{P[n]}$ in the space refer to the same vertex in the graph.

The quantity

$$l_3(P[i], P[i+1]) = \sqrt{(x_{P[i]} - x_{P[i+1]})^2 + (y_{P[i]} - y_{P[i+1]})^2 + (z_{P[i]} - z_{P[i+1]})^2}$$

is the Euclidean distance between two adjacent vertices $P[i]$ and $P[i+1]$ on path P.

The MTSP under Euclidean distances in $\mathbb{R}^d$ for any fixed $d \geq 3$ is proved to be NP-hard by Barvinok *et al.* [17]. The conventional traveling salesman problem requires that path length be minimized and the cities are defined for dimension $d = 2$. On the contrary, the MTSP requires maximization of path length and is known to be intractable in three or higher dimensions. Note that k-MARP requires maximization of $O(P)$ but is defined for $d = 2$, which makes a direct reduction from MTSP nontrivial.

Given an instance of the MTSP, we generate an instance of a k-MARP as follows. We create a graph for the k-MARP with $k = n$ using only x and y coordinates of the vertices of the MTSP (without loss of generality we assume that all coordinate values are distinct). We consider the objective

$$O(P) = \frac{\mathrm{SE}(P)}{\mathrm{PL}(P)} = \frac{\sum_{i=0}^{k-1} s_{P[i]}}{\mathrm{PL}(P)}$$

by ignoring the energy consumption component. Recall that the path loss is given by

$$\mathrm{PL}(P) = \sum_{i=0}^{k-1} 10 \log(A\, d^2_{P[i],\, P[(i+1) \bmod k]}) \tag{26.20}$$

Let $e_{P[i]}$ represent the edge between vertices $P[i]$ and $P[i+1]$ on path P. We define

$$s_{P[i]} = s(e_{P[i]}) = l_3(P[i],\ P[i+1]) + \tau[10\ \log(A d^2_{P[i],\, P[(i+1) \bmod k]})] - \alpha/k$$

A solution to the k-MARP is a path P with k hops such that

$$O(P) = \frac{\sum_{i=0}^{k-1} s_{P[i]}}{\sum_{i=0}^{k-1} d(v_{P[i]}, v_{P[i+1]})} \geq \tau$$

© 2005 by Chapman & Hall/CRC

where τ is a given nonnegative real number. After reorganizing, the objective function can be rewritten as

$$\sum_{i=0}^{k-1} \left(s(e_{P[i]} - \tau\, d(e_{P[i]})\right) = \sum_{i=0}^{k-1} (l_3(P[i],\ P[(i+1) \bmod k]) - \alpha/k) > 0 \tag{26.21}$$

which guarantees the condition necessary for a solution to the corresponding MTSP.

On the other hand, if there exists a solution to the MTSP, i.e. a closed-loop path P consisting of n edges such that $\sum_{i=0}^{n-1} l_3(P[i], P[(i+1) \bmod k]) \geq \alpha$, then this path can be used to solve the corresponding n-MARP such that $O(P) \geq \tau$. Note that the above reduction from MTSP to n-MARP is polynomial-time computable, and hence NP-hardness of the latter follows from that of the former.

The restriction of n-MARP is studied by Qi *et al.* [6], where two heuristics LCF and GCF are proposed. In the next section we propose a genetic algorithm based method for MARP and show that it outperforms the Local Closest First (LCF) and Global Closest First (GCF).

26.4 Genetic Algorithm for the MARP

26.4.1 Two-level Genetic Encoding

We design a two-level encoding scheme to adapt the GA to the MARP in an MADSN. The first level is a numerical encoding of the sensor (ID) label sequence L in the order of sensor nodes being visited by the mobile agent. For the MADSN shown in Figure 26.1, the sensor label sequence L has the following contents:

0	1	2	3	7	5	6	8	4	10	9

The first element is always set to be "0" for the reason that a mobile agent starts from the PE S_0. The mobile agent returns to S_0 from the last sensor node visited, which is not necessarily the last element of the label sequence if there are any inactive sensor nodes in the network. This sequence consists of a complete set of sensor labels because it takes part in the production of a new generation of solutions through genetic operations. It is desired to inherit as much information as possible in the new generation from the old one. For example in Figure 26.1, although nodes 3, 6, 8, and 9 are not visited in the given solution (the second-level sequence is designed to do so), they, or some of them, may likely make up a segment of a better solution than the current one in the new generation.

The second level is a binary encoding of the visit status sequence V in the same visiting order. For the MADSN in Figure 26.1, the visit status sequence V contains the following binary codes:

1	1	1	0	1	1	0	0	1	1	0

where "1" indicates 'visited' and "0" indicates 'unvisited'. The first bit corresponds to the PE and is always set to be "1" because the PE is the starting point of the itinerary. If a sensor is inactive, than its corresponding bit remains "0" until it is reactivated and visited.

Masking the first level of the numerical sensor label sequence L with the second level of the binary visit status sequence V yields a candidate path P for the mobile agent. In the above example, the path P is obtained as

0	1	2	3	7	5	6	8	4	1	9

© 2005 by Chapman & Hall/CRC

These two levels of sequences are arranged in the same visiting order for the purpose of convenient manipulations of visited/unvisited and active/inactive statuses in the implementation of the GA. The number of hops H in a path P can be easily calculated from the second level of binary sequence as follows:

$$H = \sum_{i=0}^{N} V[i] \quad \begin{cases} V[i] = 1 & \text{sensor } S_i \text{ is active and visited} \\ V[i] = 0 & \text{sensor } S_i \text{ is inactive or unvisited} \end{cases} \tag{26.22}$$

26.4.2 Implementations of Genetic Operators

We now describe the genetic operators. These operators are similar to those used in the conventional solution to the traveling salesman problem. However, we adapt the details to the current routing problem.

26.4.2.1 Selection Operator

As discussed above, the purpose of the selection operation is to select good individuals and at the same time eliminate bad individuals from the population based on the evaluation of individual fitness. In our implementation, each pair of individuals is selected randomly from the old generation to perform the crossover, mutation, and inversion operations. The fitness is computed for every newly generated child for evaluation. To maintain the same population sizes for each generation, the fitness of every newly generated child is compared with the minimum fitness of the whole population. If it is bigger than the minimum fitness value, then this child is added to the population and the individual with the minimum fitness is removed; otherwise, the new child is discarded.

26.4.2.2 Crossover Operator

We design a two-point crossover operator in our implementation for both levels of sequences. These two crossover points are selected randomly. Given the two parents

Parent 1: 1st level sequence: 0-2-7-3-/-5-1-6-/-4-9-8
2nd level sequence: 1-0-1-1-/-1-0-0-/-1-1-1
Parent 2: 1st level sequence: 0-3-5-2-/-9-6-4-/-1-7-8
2nd level sequence: 1-0-0-0-/-1-0-1-/-1-0-1

where "/" represents the crossover points, the crossover operator produces two children:

Child 1: 1st level sequence: 0-9-6-4-2-7-3-5-1-8
2nd level sequence: 1-0-1-1-/-1-0-1-/-1-1-1
Child 2: 1st level sequence: 0-5-1-6-3-2-9-4-7-8
2nd level sequence: 1-0-0-0-/-1-0-0-/-1-0-1

For the first level of label sequence, the crossover portion (between the two crossover points) of one individual is copied and inserted at the front of the other individual (immediately after label 0). All the duplicate genes in the resulting individual are knocked out to guarantee that each node appears exactly once in that individual. For the second level of visit status binary sequence, the crossover portions are simply exchanged between two individuals.

© 2005 by Chapman & Hall/CRC

26.4.2.3 Mutation Operator

We implement a two-point mutation operator that randomly selects two points and exchanges the values of these two points in both strings. As an example, consider the following parent individual:

1st level sequence: 0-2-9-3-7-1-4-5-8-6
2nd level sequence: 1-0-1-1-0-1-0-0-1-1
↑ ↑
two selected gene loci

The mutation operator produces the following child:

1st level sequence: 0-2-9-3-8-1-4-5-7-6
2nd level sequence: 1-0-1-1-1-1-0-0-0-1

26.4.2.4 Inversion Operator

We implement the inversion operator as follows. At a time, two inversion points are selected randomly to determine the inversion portion of the individual. The inversion operation is executed by reversing the order of the inversion portion of the original individual. Given one parent as follows

1st level sequence: 0-5-7-/-1-2-8-9-/-6-3-4
2nd level sequence: 1-0-1-/-1-0-1-1-/-0-0-1

where the inversion portions are enclosed by two "/" signs, the inversion operator produces the following child

1st level sequence: 0-5-7-/-9-8-2-1-/-6-3-4
2nd level sequence: 1-0-1-/-1-1-0-1-/-0-0-1

26.4.3 Parameter Selection for GAs

We usually select a high probability value above 0.9 for a genetic operator like crossover, which controls the main direction of evolution process. A low probability value below 0.1 is appropriate for genetic operators like mutation or inversion to reduce the risk of destroying the good gene segments in later generations. From experimental data, small variations in these probabilities do not have a significant impact on the performance of the GA. With respect to the maximum generation number, we select different values for different test examples in order to ensure that the optimization process approaches a steady state eventually. The difference in the best fitness values between two adjacent generations may be used as an alternative convergence indicator. In this case, the GA does not have to wait for a long time to reach the prespecified maximum generation number if the optimization process converges quickly. Its disadvantage is that the program may terminate prematurely if the optimization process does not converge quickly.

26.5 Simulation Results and Algorithm Analysis

26.5.1 Simulation Results

We compare the search results of the GA with those computed by the LCF and GCF. In most cases the LCF is able to deliver a satisfying route for a mobile agent; hence, it is a comparable algorithm to the

© 2005 by Chapman & Hall/CRC

GA. The GCF may find a path with a lesser number of hops than the LCF, but it usually has a significantly longer path length, resulting in unacceptable path loss. A series of experimental networks of different sensor node sizes and distribution patterns are created to conduct the optimal routing. The spatial locations of all the nodes are randomly selected. The LCF and GCF algorithms pick up the center node as the starting point in each network. About 1–10% of the sensors are shut down uniformly over the surveillance region. All sensor parameters for data acquisition and the wireless channel in the MADSN use the real-life data from the field demo listed in Table 26.1.

In order to make a visual comparison, the search results computed by the GA, LCF, and GCF for the first relatively small sensor network are shown in Figure 26.2, Figure 26.3, and Figure 26.4 respectively. This sensor network consists of 200 nodes, eight of which are in the sleep state. A quantified amount of

Table 26.1. The MADSN parameters

Sensor node processor type	Hitachi SuperH processor SH-4 architecture
Sensor node processor speed	200 MHz
Mobile agent sizes	400 bytes
Ave. data sizes	100 bytes
Carrier frequency band	2.4 GHz
Transmitting power	100 mW
Transmitter gain	2
Receiving power	80 mW
Receiver gain	2
Channel operation frequency	20 kHz
Channel width	16 bits
Data sampling rate	20 kHz
Sample data format	16-bit

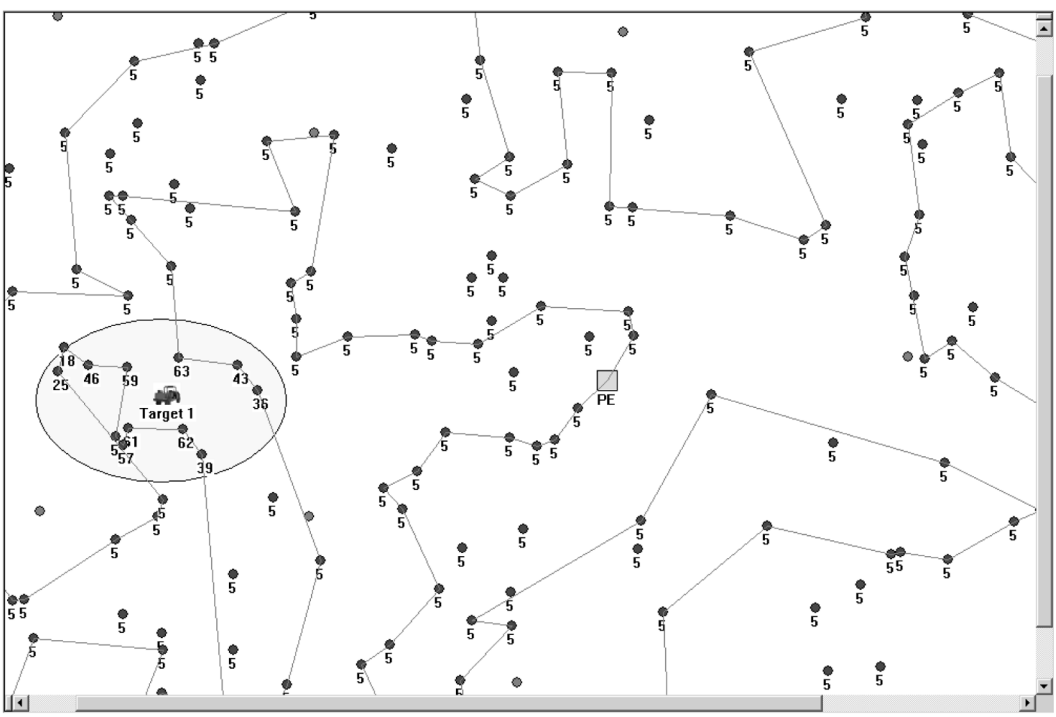


Figure 26.2. Visualization of the search result computed by the GA for an MADSN with 200 nodes.

© 2005 by Chapman & Hall/CRC

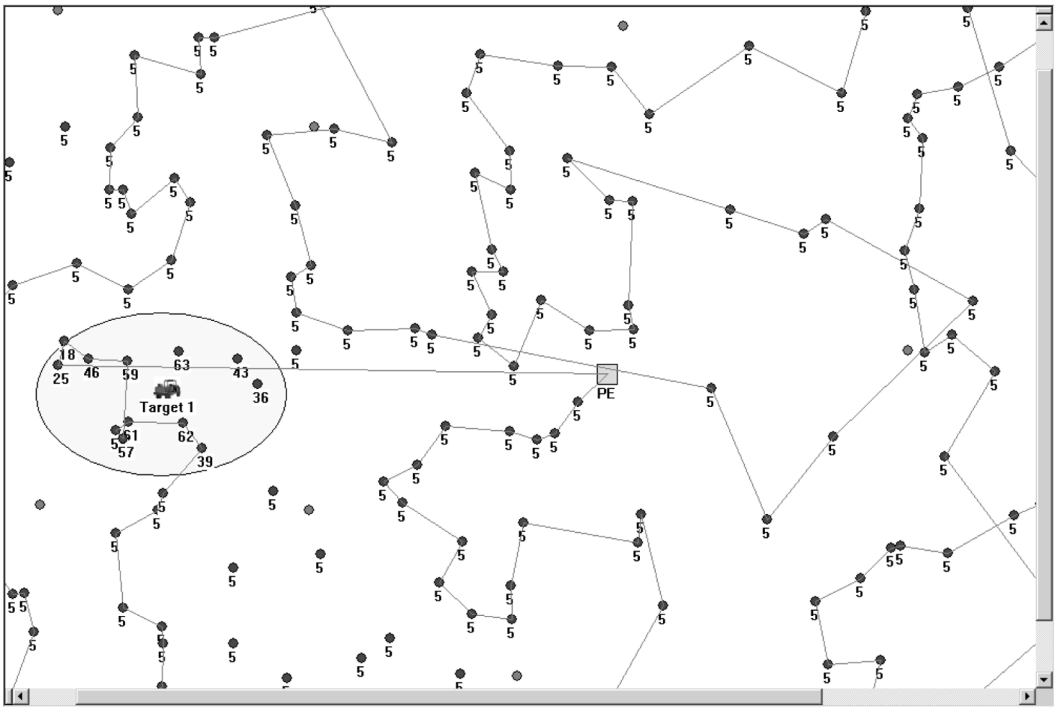


Figure 26.3. Visualization of the search result computed by the LCF for an MADSN with 200 nodes.

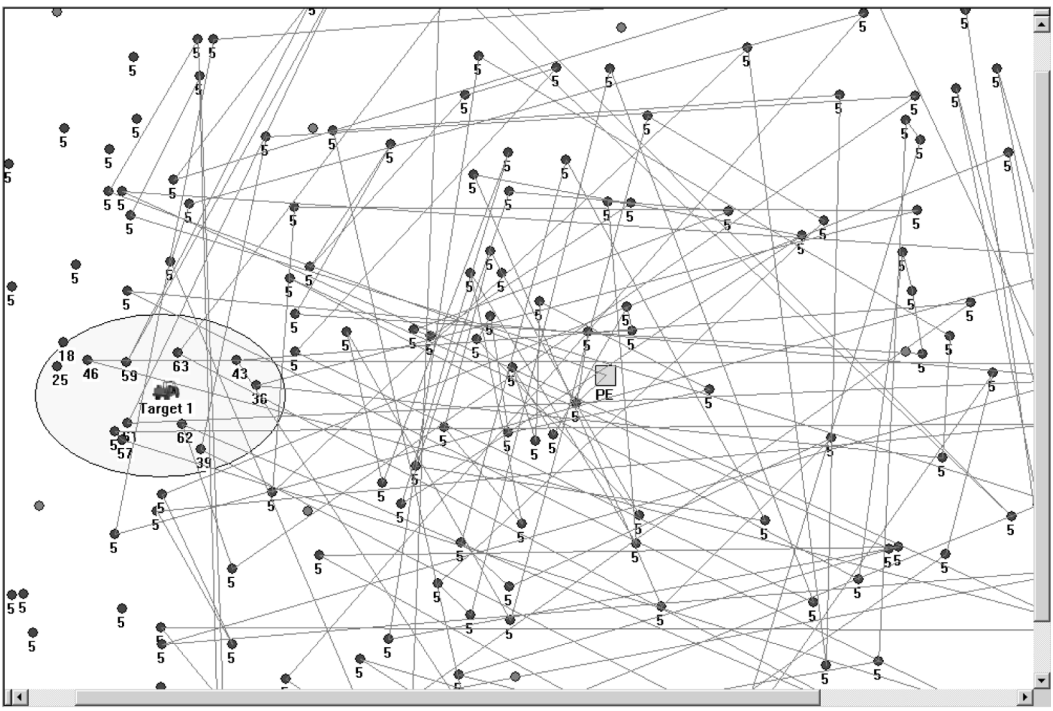


Figure 26.4. Visualization of the search result computed by the GCF for an MADSN with 200 nodes.

© 2005 by Chapman & Hall/CRC

signal energy associated with each active sensor, ranging from 0 to 64, is displayed under the corresponding sensor node. The minimum acceptable amount of signal energy detected by an individual sensor node is 5, and inactive nodes do not detect any signal energy. There is one potential target located in the region. The sensor nodes in the vicinity of the targets detect a higher signal energy than the other nodes. The total signal energy detected is 1467 units, and the acceptable signal level for correct inference is set to be 1200 units. A maximum generation number if 200 is specified for the GA as the convergence indicator, which informs the program when to stop the searching process.

It has been observed that the optimization process of the GA moves forward rapidly in the beginning, and becomes slow and stable in the later stages of computation, especially after the generation number reaches 100.

Table 26.2 shows that the GA uses 140 hops to achieve an acceptable signal level, whereas the LCF uses 168 hops and the GCF uses 147 hops. The path losses of the GA, LCF and GCF are 22,919 units, 26,864 units, and 28,748 units respectively. The migration of the mobile agent along the path computed by the GA consumes 407 units of energy, the LCF consumes 489 units of energy, and the GCF consumes 428 units of energy. The simulation results of other sensor networks with larger node sizes and randomized distribution patterns are also tabulated in Table 26.2.

In Table 26.2, the total signal energy detected represents the maximum energy detected by all active sensors deployed in the region under surveillance. The acceptable signal energy level is a given value desired for a specific application.

Figures 26.5–26.8 illustrate the corresponding curves of node sizes versus number of hops, node sizes versus path loss, node sizes versus energy consumption, and node sizes versus objective values, respectively. From Table 26.2 and Figures 26.5–26.7, it has been seen that in most cases the GA is able to find a satisfying path with less number of hops, less energy consumption, and less path loss than the LCF and GCF algorithms. Figure 26.8 clearly shows that the GA has a superior overall performance over the two other heuristics in terms of the objective function defined in this implementation.

The current GA program was implemented in C++ using MFC with GUI and per-generation result display. The code takes a few seconds to run the first 100 generations for a network of hundreds of nodes. The GA runs much faster and its executable code size decreases significantly when a GUI is not implemented. In such cases, the GA run time may not be a serious problem for semi-dynamic routing, which will be discussed next.

26.5.2 Algorithm Comparison and Discussion

The GCF algorithm is relatively simple and fast but suffers from poor performance in terms of path loss. The GCF algorithm essentially utilizes sorting to compute the path. Its computational complexity is $O(N \log N)$ if using a comparison-based sorting algorithm. The LCF algorithm has the computation complexity of $O(N^2)$ if the closest neighbor node is obtained by simple comparison in each step. The analysis for the computation complexity GA is more complicated. After making some simplifications in its implementation, the computation complexity for the genetic algorithm is $O(NMG)$, where N is the number of nodes in the network, M is the initial population size, and G is the maximum generation number used to indicate the end of the computation.

As is the case for the GCF algorithm, the performance of the LCF algorithm also depends significantly on the network structure [6]. In some bad cases, it may end up with unacceptable results. Comparatively, the network structure has much less influence on the performance of the GA owing to its random search technique.

Unlike the LCF and GCF algorithms, it is not necessary to specify a starting node for the implementation of the GA. Actually, any active sensor node can be designated as a starting node in the GA, whereas the performance of the LCF and GCF algorithms are crucially dependent on the location of the starting node. In addition, no matter in what order the nodes are visited, the GA always comes up with a closed route for the mobile agent to come back to its starting node.

© 2005 by Chapman & Hall/CRC

Table 26.2. Comparisons of search results of the GA, LCF, and GCF for networks of different node sizes and distribution patterns

Case #	Node size	No. of dead sensors	No. of potential targets	Total signal energy detected	Acceptable signal energy level
1	200	8	1	1,467	1,200
2	300	4	2	2,985	2,750
3	400	5	2	4,000	3,680
4	500	8	3	4,080	3,710
5	600	9	4	4,980	4,800
6	700	10	5	5,340	5,190
7	800	11	4	6,200	5,950
8	900	13	3	7,000	6,380
9	1000	14	5	7,525	7,000
10	1100	13	4	8,515	7,990
11	1200	18	5	10,050	9,425
12	1300	16	4	11,410	9,600
13	1400	15	4	11,500	9,800
14	1500	19	5	12,380	11,000
15	1600	25	6	13,505	12,210

Case #	GA				LCF				GCF			
	No. hops	Path loss	Energy consumption	Achieved signal energy	No. hops	Path loss	Energy consumption	Achieved signal energy	No. hops	Path loss	Energy consumption	Achieved signal energy
1	140	22,919	407	1,202	168	26,864	489	1,215	147	28,748	428	1,204
2	216	34,154	670	2,785	273	42,027	796	2,757	228	43,191	664	2,763
3	300	47,100	875	3,691	379	57,842	1106	3,684	291	55,897	849	3,688
4	424	64,302	1237	3,716	460	68,645	1342	3,721	424	82,817	1237	3,714
5	483	76,205	1389	4,815	561	82,627	1637	4,807	490	95,417	1430	4,819
6	544	88,344	1552	5,195	663	96,370	1935	5,197	566	110,972	1652	5,194
7	613	98,722	1803	5,963	757	109,877	2186	5,952	708	139,179	2093	5,958
8	703	113,790	2079	6,400	847	121,082	2473	6,390	774	151,392	2260	6,385
9	838	122,087	2481	7,001	948	134,053	2768	7,003	929	183,379	2713	7,004
10	931	137,906	2704	8,002	1059	150,332	3123	7,990	988	207,397	2845	8,005
11	1067	152,397	3116	9,425	1146	161,248	3347	9,460	1117	221,073	3262	9,450
12	1009	145,899	2946	9,650	1208	171,032	3528	9,600	1138	231,898	3301	9,610
13	1105	158,471	3227	9,840	1287	175,492	3693	9,800	1174	239,843	3459	9,820
14	1270	170,029	3634	11,005	1381	192,300	4099	11,019	1315	258,942	3832	11,004
15	1369	182,382	3823	12,210	1534	211,317	4480	12,210	1495	295,711	4366	12,240

Case #	GA objective value $O(P) = \mathrm{SE}(P)(1/\mathrm{EC}(P)+1/\mathrm{PL}(P))$	LCF objective value $O(P) = \mathrm{SE}(P)(1/\mathrm{EC}(P)+1/\mathrm{PL}(P))$	GCF objective value $O(P) = \mathrm{SE}(P)(1/\mathrm{EC}(P)+1/\mathrm{PL}(P))$
1	3.005763	2.52989	2.854965
2	4.238259	3.529169	4.225116
3	4.296651	3.394613	4.409913
4	3.061832	2.826934	3.047271
5	3.529708	2.994646	3.420435
6	3.406098	2.739716	3.190872
7	3.367668	2.776951	2.88944
8	3.134647	2.63668	2.867397
9	2.87919	2.582226	2.619838
10	3.017345	2.611586	2.852306
11	3.086556	2.885079	2.939742
12	3.34177	2.777218	2.95268
13	3.111365	2.709512	2.879914
14	3.093068	2.745518	2.914104
15	3.260774	2.783227	2.844873

© 2005 by Chapman & Hall/CRC

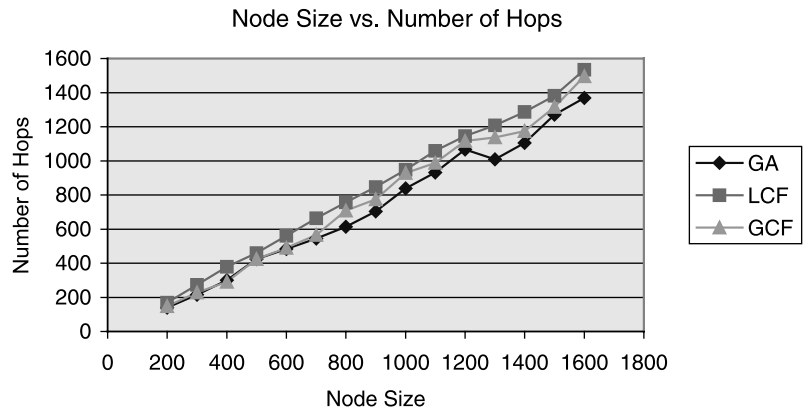


Figure 26.5. Node sizes versus number of hops for the three algorithms.

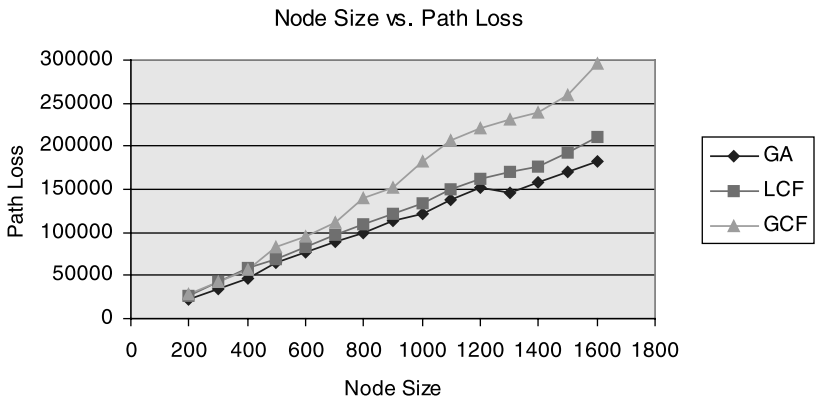


Figure 26.6. Node sizes versus path loss for the three algorithms.

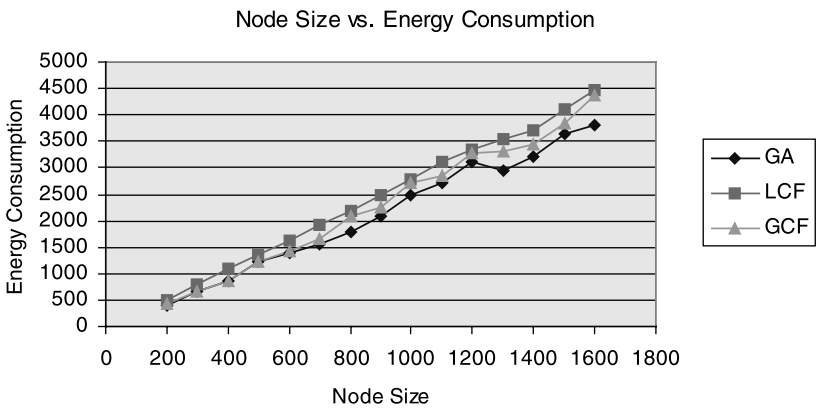


Figure 26.7. Node sizes versus energy consumption for the three algorithms.

The mobile agent routing algorithm can be classified as dynamic and static routing according to the place where routing decisions are taken. A dynamic method determines the route locally, on the fly, at each hop of the migration of a mobile agent among sensor nodes. A static method uses centralized routing, which computes the route at the PE node in advance of mobile agent migration. For different

© 2005 by Chapman & Hall/CRC

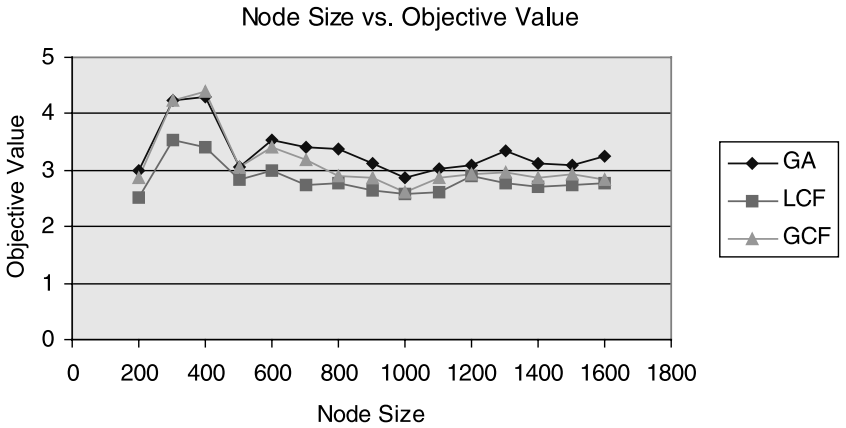


Figure 26.8. Node sizes versus objective values for the three algorithms.

sensor network applications, either dynamic or static method can be applied. For example, it might be sufficient to use a static routing for target classification, but target tracking may require a dynamic routing due to its real-time constraint.

The LCF algorithm is suitable for carrying out dynamic routing because each step of its computation depends only on the location of the current node, whereas the GCF is in favor of static routing, whose computation can be carried out offline based on the global network structure. Both LCF and GCF are deterministic routing, and always supply the same path between a source–destination pair in a given network. The GA collects information about the network status from all sensor nodes so that it is able to conduct adaptive routing. However, broadcasting the detected signal energy produces extra communication overhead.

Since it is desired to keep the mobile agent code as compact as possible, the GA may be used to implement a semi-dynamic routing. In this routing scheme, the routing code does not go with the mobile agent. If the network system is notified of some events (e.g. some nodes are shut down or activated, or do not have enough energy remaining to transmit the signal along the previously designated link), which causes the previously computed route to be invalid, then the routing code is rerun based on the updated system information, and the new route is transmitted to the mobile agent for its further migration.

This semi-dynamic routing scheme is supported by the robustness of the sensor network. According to the experience from the field demo, sensor nodes usually function well once brought up, and the network may remain stable continuously for 1–2 h of sessions.

26.6 Conclusions

We presented a mobile-agent-based paradigm for data fusion in DSN. By utilizing a simplified analytical model for the DSN, we formulated a route computation problem for the mobile agent in terms of maximizing the received signal strength while keeping path loss and energy consumption low. This route computation problem turned out to be NP-hard, thereby making it highly unlikely to develop a polynomial-time algorithm, to compute an optimal route. Hence, we proposed a GA to solve this problem by employing a two-level genetic encoding and suitable genetic operators. Simulation results are presented for comparison between our GA and existing LCF and GCF heuristics. Various aspects of the proposed algorithm, such as computational complexity, impact of network structure and starting node, and dynamic and static routing, are discussed.

Future research work is to be focused on exploring more complex routing models with more general objective functions. For example, in the current model we assume that the sensor locations are fixed

© 2005 by Chapman & Hall/CRC

once they are manually deployed, which is the case in the field demo. However, in a real-world sensor network, sensors could be airborne or installed on vehicles or robots. The mobility of sensors brings new challenges to the design of dynamic routing algorithms for mobile agents. In addition, instead of using the simple free-space propagation model to compute path loss, more complex empirical propagation models may be studied and applied in the construction of objective function.

Acknowlegment

This research is sponsored by the Material Science and Engineering Division, Office of Basic Energy Sciences, U.S. Department of Energy, under Contract No. DE-AC05-00OR22725 with UT-Battelle, LLC, the Defense Advanced Projects Research Agency under MIPR No. K153, and by National Science Foundation under Grants No. ANI-0229969 and No. ANI-335185.

References

[1] Hyder, A.K. *et al.*, *Multisensor Fusion*, Kluwer Academic Publishers, 2002.
[2] Iyengar, S.S. and Wu, Q., Computational aspects of distributed sensor networks, in *Proceedings of International Symposium on Parallel Architectures*, *Algorithms and Networks*, May 22–24, Manila/ Makati, Philippines, IEEE Computer Society Press, 2002.
[3] Jayasimha, D.N. and Iyengar, S.S., Information integration and synchronization in distributed sensor networks, *IEEE Transactions on Systems*, *Man, and Cybernetics*, 21(5), 1032, 1991.
[4] Zheng, Y.F., Integration of multiple sensors into a robotics system and its performance evaluation, *IEEE Transactions on Robotic Automation*, 5, 658, 1989.
[5] Luo, R.C. and Kay, M.G., Multisensor integration and fusion in intelligent systems, *IEEE Transactions on System*, *Man*, *and Cybernetics*, 19, 901, 1989.
[6] Qi, H. *et al.*, Multi-resolution data integration using mobile agents in distributed sensor networks, *IEEE Transactions on Systems, Man, and Cybernetics Part C: Applications and Reviews*, 31(3), 383, 2001.
[7] Holland, J.H., *Adaptation in Nature and Artificial Systems*, The University of Michigan Press, 1975. (Reprinted by MIT Press, 1992.)
[8] Coley, D.A., *An Introduction to Genetic Algorithms for Scientists and Engineers*, World Scientific, 1999.
[9] Winston, P.H., *Artificial Intelligence*, 3rd ed. Addison-Wesley Publishing Company, 1993.
[10] Chen, G., *Genetic Algorithm and Its Applications*, People's Post Publishing House, China, 1996.
[11] Goldberg, D.E., *Genetic Algorithms in Search, Optimization & Machine Learning*, Addison-Wesley Publishing Company, 1989.
[12] Iyengar, S.S. *et al.*, A versatile architecture for the distributed sensor integration problem, *IEEE Transactions on Computers*, 43(2), 175, 1994.
[13] Rev. A, *WINS NG 2.0 User's Manual and API Specification*, Sensoria Corporation, May 30, 2002.
[14] Rappaport, T.S., *Wireless Communications Principles and Practice*, 2nd ed., Prentice Hall PTR, 2002.
[15] Rao, N.S.V., Multisensor fusion under unknown distributions: finite sample performance guarantees, in *Multisensor Fusion*, Hyder, A.K. *et al.* (eds), Kluwer Academic, 2002.
[16] Swaminathan, V. and Chakrabarty, K., Real-time task scheduling for energy-aware embedded systems, in *IEEE Real-Time Systems Symposium* (*Work-In-Progress Sessions*), Orlando, FL, November 2000.
[17] Barvinok, A. *et al.,* The geometric maximum traveling salesman problem, on the maximum TSP. *Journal of the ACM* 50(5), 641, 2003.

© 2005 by Chapman & Hall/CRC

Appendix A

Class definitions of abstract sensor, leaf node, processing element, and wireless link used in the algorithm implementations are listed as follows:

Definition of sensor class *Csensor*

```
class CSensor
{
  unsigned int   m_sensorLabel;  // a unique sensor ID: 0 for PE, else for leaf nodes
  bool           m_sensorStatus;  // TRUE: active; FALSE: inactive
  double         m_processorSpeed;
  double         m_locationLongitude
  double         m_locationLatitude;
  double         m_dataProcessingTime;
  double         m_transmittedPower;
  double         m_transmitterGain;
  double         m_receivedPower;
  double         m_receiverGain;
}
```

Leaf node class derived from *CSensor*

```
class CLeafNode: public CSensor
{
  bool      m_visited;
  double    m_dataAcquisitionTime;
  double    m_detectedSignalEnergy;
  double    m_dataSamplingRate;
  double    m_sampleDataFormat;
}
```

Processing element class derived from *CSensor*

```
class CProcessingElement: public CSensor
  {
    double   m_setupTime;
  }
```

© 2005 by Chapman & Hall/CRC

Definition of wireless link class *CLink*

```
class CLink:
  {
    CSensor*      m_pSensorTransmitter;
    CSensor*      m_pSensorReceiver;
    double        m_linkDistance;
    double        m_bandWidth;
    double        m_channelWidth;
    double        m_operateFrequency;
    double        m_carrierFrequency;
    double        m_linkPropagationTime;
    double        m_msgTransmissionTime;
    double        m_linkPowerLoss;
    double        m_systemLossFactor;
  }
```

Most of the attributes defined in these classes are self-explanatory. Leaf node and processing element classes are, in turn, derived from the abstract sensor class *CSensor*. In the definition of leaf node class, the attribute *m_dataSamplingRate* represents the frequency at which the signal data is sampled. The amount of memory space used to store the sampled data is determined by the attribute *m_sampleDataFormat*. In the definition of wireless link class, *m_systemLossFactor* is a parameter of the free-space propagation model, which is not necessarily related to the physical propagation.

© 2005 by Chapman & Hall/CRC

27

Computer Network — Basic Principles

Suresh Rai

27.1 Introduction

Today, the word computer network is a synonym for Internet or information super-highway and is a household name. Its mass appeal among novices, nerds, and pundits is primarily due to the applications like electronic mail (e-mail), the World Wide Web (www), remote terminal access (telnet), and different protocols such as file transfer protocol (ftp), network file system (NFS), network news transfer protocol (NNTP), etc. These applications have made information dissemination easy, timely, and cool. Not long ago, when the Mars Path Finder Lander (nicknamed Sagan Memorial Station) landed on the surface of Mars on July 4, 1997 (Independence Day for the USA), almost everybody was surprised from the interest of a vast number of people who wanted to know the results and look at the first-ever high-resolution color images of the Martian surface themselves. This 'get-self-involved' urge took them not in front of TV sets to watch a reporter narrate the story, but to the Internet, where they felt satisfied by watching the story revealing itself through images that were posted on the Internet almost instantaneously by NASA scientists. More than one million people visited the Jet Propulsion Laboratory Webpage and its various mirror sites created for this purpose using their Web browsers and service providers;[1] this was a record in itself. Since 1997, the popularity of the Internet has grown exponentially. It is a worldwide collection of more than 250,000 networks, public and private, that have agreed to use common protocols and exchange traffic. An internet (lowercase "i") or internetwork refers to a set of networks connected by routers and appears to its users as a single network. A network can be taken as a set of machines or hosts linked by repeaters or bridges.

27.2 Layered Architecture and Network Components

The basic objective of a computer network is for an application on one node to communicate with another application or device on another node. An application could be a file transfer, terminal access, e-mail, resource sharing, etc. While this may sound simple, some complexities are involved with the

[1]Some typical service providers include America On Line (AOL), Juno On Line, CompuServe, AT&T, MCI, etc.

© 2005 by Chapman & Hall/CRC

possibility of different implementations of wide area network (WAN), metropolitan area network (MAN), and local area network (LAN) systems. This section, first, discusses a reference model from the International Standards Organization (ISO) and, then, introduces the Transmission Control Protocol/Internet Protocol (TCP/IP) architecture (commonly known as the Internet architecture). We also describe some typical internetworking components, such as repeater, bridge, and router.

27.2.1 Layering and OSI Model

The basic purpose of layering is to separate network-specific functions to help make the implementation transparent to other components. Thus, elements of layered specifications allow standard interfaces for "plug-and-play" compatibility and multi-vendor integration. Other reasons for having layered systems include enhancing and information-hiding features, ease of modification and testing, and portability. As an example of adding features or facilities, consider an unreliable physical layer that is made reliable through the use of a data link layer supporting an automatic repeat request (ARQ) scheme. Finally, layering allows us to reuse functionality. This is typically evident with an operating system design where a lower layer implements functionality once, which can then be shared by many upper layer application programs.

Layering is a form of "information hiding," which offers advantages but also leads to some problems. A lower layer provides a service interface to an upper layer, hiding the details of "how is it done"? Further, it is sometimes difficult to allow efficient performance out of a layer without violating the layer boundary. As an example, consider the protocol stack with error and flow control functions. The error control feature deals with the ability to retransmit packets corrupted on a link; the flow control, on the other hand, relates to the rate at which packets are placed by the source on the link. A flow control that relies on network congestion and offers a better job from the performance viewpoint poses a threat to layer boundary violation. This is because the flow control layer needs to know the details of data transfer over the local links in the network. To obtain an efficient performance, it becomes imperative to "leak" enough information between layers. Choosing a balance between the "leak" and "information hiding" feature of the layer model provides a hallmark of good protocol stack design.

Most communication environments use layering to separate the communication functions and application processing. For example, the OSI[2] reference model divides the communication between any two networked computing devices into seven layers or categories. Commonly used architectures, such as AppleTalk and TCP/IP, on the other hand, have only five layers. It is important to remember that a layered model does not constrain an implementation; it provides a framework. Implementations, thus, do not conform to the OSI reference model, but they do conform to the standards developed from the OSI reference.

In the following, we consider a top-down approach to describe seven OSI layers and their functionality. Layer 7 (1) stands for the top (bottom)layer.

Layer 7 (*application layer*). This layer contains the programs that perform the tasks desired by users. Such tasks may include file transfer, e-mail, printer service, remote job entry, and resource allocation. Special-purpose applications, such as Gopher, Fetch, and Wide Area Information Server (WAIS), help navigate the way to resources on the Internet. Similarly, the World Wide Web links thousands of servers using a variety of formats, including text, graphics, video, and sound. Browsers such as Netscape and Internet Explorer are used for the purpose.

Layer 6 (*presentation layer*). The presentation layer accepts a message from the application layer; provides formatting functions such as encryption, code conversion, and data compression; and then

[2]The ISO Open System Interconnection (OSI) reference model is termed *open* because, unlike proprietary architectures such as IBM's System Network Architecture (SNA) and Digital Equipment Corporation's DECnet, its details are publicly available to anyone at little or no charge.

© 2005 by Chapman & Hall/CRC

passes the message to the session layer. Encryption is needed for security reasons. Note that data can be represented by any of several different codes, the two most common being American Standard Code for Information Interchange (ASCII) and Extended Binary Coded Decimal Interchange Code (EBCDIC). Also, different computers store data differently in the memory. The big (little) endian strategy considers high-order (low-order) byte first in the memory. Code conversion is essential to take care of all these variations. Other Layer 6 standards guide graphic and visual presentation. PICT is a picture format used to transfer QuickDraw graphics between Power PC or Macintosh programs. Tagged Image File Format (TIFF) is a standard graphics format for high-resolution, bit-mapped images. JPEG standards come from the Joint Photographic Experts Group. For sound and movies, presentation layer standards include MIDI (Musical Instrument Digital Interface) for digitized music and the MPEG (Motion Pictures Expert Group) standard for compression and coding of motion video for CDs and digital storage.

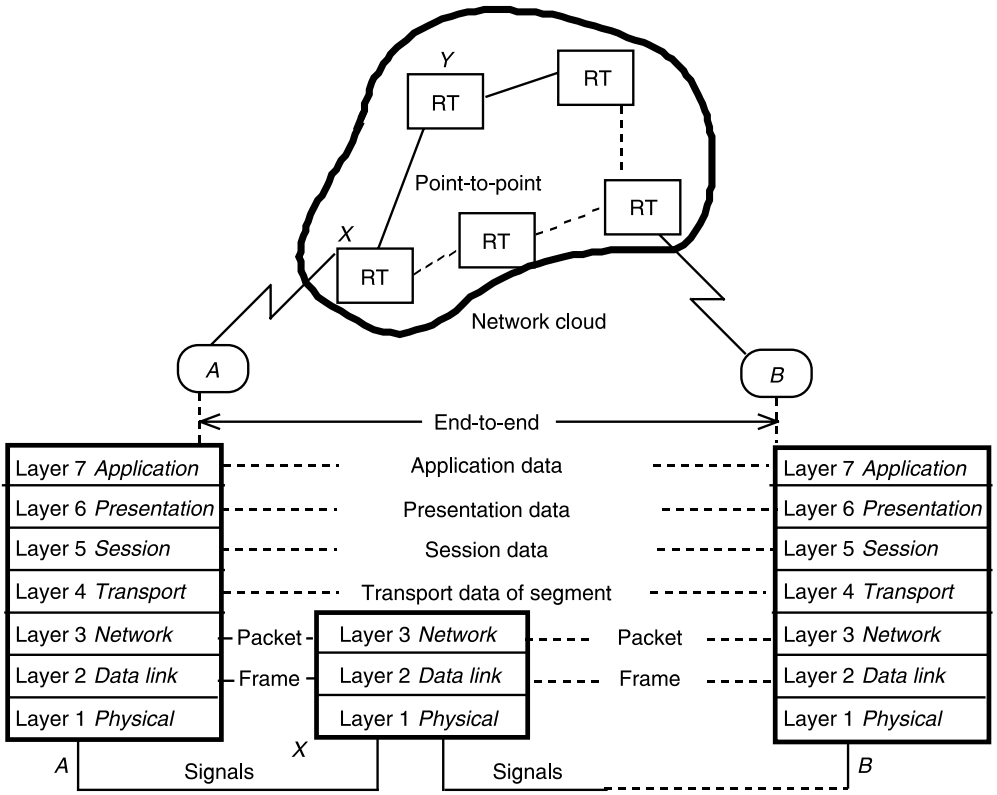
Example 27.1. To illustrate the need for MPEG data compression, consider a frame for network video having 352×240 pixels (a pixel refers to one dot on the display). If we use 24 bits per pixel (to help maintain the color), each frame will necessitate 247.5 Kbytes. To make it video quality, one requires a rate of 25–30 frames/s, which produces 247.5×30 Kbytes/s or approximately 60 Mbps. Obviously, some method such as the MPEG technique is needed to compress the bit stream to reduce its rate to 1.5 Mbps. Equipping multimedia clients and servers with compression/decompression capability will, thus, lower the demand on the medium.

Layer 5 (*session layer*). This layer enables two applications to establish, manage, and terminate a session (or logical connection) to facilitate communication across the network on an end-to-end basis (refer to Figure 27.1). Essentially, the session layer coordinates service requests that occur when applications communicate between different hosts. For example, a user may ''logon'' to a remote system and may communicate by alternately sending and receiving messages. The session layer helps coordinate the process by informing each end when it can send or must listen. This is a form of synchronization. The session layer also controls data flow (which can be either full or half duplex) and provides recovery if a failure occurs. As an example, let a user be sending the contents of a large file over a network that suddenly fails. Instead of retransmitting the file from the beginning, the session layer allows the user to insert checkpoints in a long stream. If a network crashes, then only the data transmitted since the last checkpoint is lost. The following are typical examples of session-layer protocols and interfaces:

- NFS, developed by Sun Microsystems, allows transparent access to remote network-based resources. It is used with TCP/IP and UNIX.
- Remote Procedure Call (RPC) provides a general redirection mechanism for distributed service environments. RPC procedures are built on clients, and then executed on servers.
- AppleTalk Session Protocol (ASP) is used to establish and maintain sessions between an AppleTalk client and a server.
- X-Window system permits intelligent terminals to communicate with remote UNIX computers as if they were directly attached monitors.

Layer 4 (*transport layer*). The fourth layer arranges for end-to-end delivery of messages between transport service access points (TSAPs) or ports. For this, a transport layer often fragments messages into segments and reassembles them into messages, possibly after resequencing them. (A layer 4 packet is called ''segment.'') It also oversees data integrity. In other words, layer 4 ensures that no messages are lost or duplicated and that messages are free of errors. As an example, a typical protocol in this layer may use TSAPs of the source and destination session entities, together with a checksum to detect errors, and message sequence numbers to ensure that messages are successfully received and are in order. Further, the transport layer delivery of messages is either *connection-oriented* or *connectionless*. A connection-oriented service provides a reliable transport and is modeled after the telephone system. Its two typical examples include remote login and digitized voice. In such cases, the communicating end

© 2005 by Chapman & Hall/CRC



(a)

	Wide Area Network, WAN	Local Area Network, LAN			
Data link layer	SDLC, HDLC, X.25, Frame Relay, ISDN, PPP	LLC (802.2)			Ethernet
Physical layer	V.24, EIA-232	802.3	802.5	FDDI	

(b)

Figure 27.1. (a) A network cloud, scope of end-to-end and point-to-point communication, and OSI layers (RT means a router). (b) Physical and data link standards.

systems accomplish the following:

- Ensure that segments delivered will be acknowledged back to the sender.
- Provide the retransmission of any segments that are not acknowledged.
- Put segments back into their correct sequence at the destination.
- Provide congestion avoidance and control.

Contrary to this, a service such as electronic (junk) mail does not require connections. All that is needed is a best effort strategy, i.e. a high probability of arrival, but no guarantee. It is achieved through a connectionless service that is modeled after the postal system

Layer 3 (*network layer*). The network layer supports naming, addressing, accounting, and routing functions. Naming provides a way to identify a host in the network and is mainly for human usage.

© 2005 by Chapman & Hall/CRC

The addressing at layer 3 provides an alternative means to locate computer systems on an internetwork and is primarily used by the machine or computer. A name server then resolves between the host name and its (logical) address. There are several addressing schemes, depending on the protocol family being used. In other words, AppleTalk addressing is different from TCP/IP addressing, which in turn is different from OSI addressing, and so on. Regardless of the protocol type, a network layer address is generally called a virtual or logical address and is often hierarchical. This greatly enhances which addresses could be assigned and also helps enable scaleable aggregation of addressing and routing information. Upon receiving a packet from the transport layer, the network layer logs the event to the accounting system and then prepares the packet for transmission to the next node on the path to the destination. It does this by looking up the destination address in its network routing table to find the next address along the path. (This approach is called destination-based routing.) The function — finding the path the packets must follow — is called routing.[3] A proper routing strategy, such as routing information protocol (RIP), open shortest path first (OSPF), border gateway protocol (BGP), and protocol independent multicasting (PIM) helps build the routing table that provides a fastest, cheapest, or safest path.

Layer 2 (*data link layer*). This layer is responsible for the transmission of a frame on point-to-point links or between two adjacent nodes (see Figure 27.1). Typically, a frame encapsulates a packet by appending additional fields such as a unique hardware[4] (physical or link layer) address of the network interface card (NIC) that identifies the machine attached to a shared link (technically, a physical address is not needed for a point-to-point link), cyclic redundancy check (CRC) bits to accomplish low-level transmission error detection, and control fields to help perform error recovery and flow control. In this layer, we also consider methods such as bit stuffing, byte stuffing, or code violation for achieving data transparency, which refers to the ability of the data link to transmit any bit combination (arising from binary or text file). Figure 27.1 shows standards for the data link layer.

Layer 1 (*physical layer*). The physical layer pertains to the transmission of bits of data over a physical medium, such as twisted pair wire, cable, optical fiber, or satellite link and hence characterizes the transmission medium, the nature of the signals, the data rate, and related matters. It also defines the mechanical, electrical, functional, and procedural aspects of the interface between the data terminal equipment (DTE), such as a computer or workstation, and data circuit-terminating equipment (DCE), such as a modem. For example, a commonly used physical layer interface in the U.S. is EIA-232-D. Its mechanical specifications include the physical dimensions, latching and mounting arrangements, and so forth of its D-shaped 25 pin interface connector. The electrical characteristics provide voltage levels and timing (such as pulse rise times and duration). Functional specifications assign meanings to circuits or pins, which are divided into four subgroups: data, control, timing (bit and byte), and grounds. Finally, procedural sequences provide handshaking rules to set up, maintain, and deactivate physical-level interconnections that help accomplish data transfer.

[3]In fact, network layer routing has two components: forwarding and control. The forwarding component deals with actual forwarding of packets and works on a destination basis, i.e. the decision about where to forward a packet is made based only on its destination address. The control component is responsible for the construction and maintenance of the forwarding (routing) table and consists of one or more routing protocols.

[4]Contrary to logical address, a hardware address is flat. One good example of a flat address space is the U.S. social security numbering system, where each person has a single, unique social security number. The hardware address is unique for each network connection. The NIC is the hardware in the computer that enables one to connect it to a network. On most LAN-interface cards, the physical address is burned into ROM; when the NIC is initiated, this address is copied into RAM. A host or computer has only one NIC because most computer systems have one physical network connection; they have only a single link-layer address. Routers and other systems connected to multiple physical networks can have multiple link-layer addresses. Such a machine, having more than one network interface, is called a multi-homed host (MH). However, an MH does not work as a router (which always has more than one NIC). Some typical examples of the MH include servers such as NFS, database, and firewall gateways, because all these are configured as MH hosts.

© 2005 by Chapman & Hall/CRC

27.2.2 TCP/IP Layering

The TCP/IP suite was developed as part of the research done by the DARPA. Later, TCP/IP was included with the Berkeley Software Distribution of UNIX. The Internet protocol suite includes not only layer 3 and 4 specifications (such as IP and TCP), but also specifications for such common applications as e-mail, remote login, terminal emulation, and file transfer. Loosely, the TCP/IP refers to the Internet architecture and upholds only five layers of the OSI reference model (Table 27.1).

The application layer in TCP/IP combines the features of layers 5 through to 7 of the OSI model and supports protocols such as file transfer, e-mail, remote login, etc. Network management is also an important ingredient of this layer. As is obvious from Table 27.1, the TCP/IP stack maps closely to the OSI reference model in layers 4 and 3. TCP and user datagram protocol (UDP) represent two protocol examples working at level 4. The transport layer of TCP performs two functions, namely flow control and reliability. The flow control is provided by sliding windows and the reliability is achieved through sequence numbers and acknowledgments. As mentioned earlier, the UDP is a connectionless protocol and uses no windowing or acknowledgments. In this case, an application layer protocol such as Trivial FTP, Simple Network Management Protocol, Network File Server, and Domain Name System is responsible for providing the reliability. The network layer of TCP/IP contains following protocols:

- IP provides connectionless, best effort delivery routing of datagrams. Note that a packet at layer 3 is called a datagram. Further, the IP layer is not concerned with the content of the datagrams. It looks for a way to move the datagrams to their destination.
- Internet Message Control Protocol (ICMP) provides control and messaging capabilities and is implemented by all TCP/IP hosts. An ICMP message is carried in an IP datagram.
- Address Resolution Protocol (ARP) determines the data link layer address for a known IP address. Reverse ARP (RARP), on the other hand, obtains network addresses when data link layer addresses are known. ARP and RARP are implemented directly on top of the data link layer, especially over a multi-access medium such as Ethernet.

The network interface at layers 2 and 1 are technically not defined as part of the TCP/IP stack. Hence, all the variations given in Figure 27.1(b) can be applied for this interface, which could be a LAN or Point-to-Point Protocol (PPP) connection. For example, Figure 27.2(a) shows two hosts, A and B, connected to an Ethernet segment. Here, a model illustrating layers at each node explains the basic concepts including the peer (or similar) protocols. Thus, the TCP at A has a peer relationship with the TCP at B and so do the IPs at both hosts. Note that peers communicate to each other through entities supported by their lower levels. To explain this, we have shown the encapsulation, or how application data is packaged for transmission. As is obvious from Figure 27.2(b), an application's data is handled by TCP/UDP as a segment, in which the corresponding transport layer header is attached to the data. The network layer, in turn, takes the segment and passes it as a datagram to the data link layer after

Table 27.1. Internet protocol suite versus OSI layers

Layer	OSI	Internet or TCP/IP suite		
		TCP/IP	Data format	Protocols
7	Application	Application	Messages	Telnet, ftp, TFTP, SMTP,
6	Presentation		or streams	SNMP, etc.
5	Session			
4	Transport	Transport	Segment	TCP, UDP
3	Network	Network	Datagram	IP
2	Data link	Data link	Frame	PPP[a]
1	Physical	Physical	Bits	

[a]Besides PPP, the data link layer also supports other layers depending on the type of networking hardware being used (Ethernet, token ring, FDDI, etc.).

© 2005 by Chapman & Hall/CRC

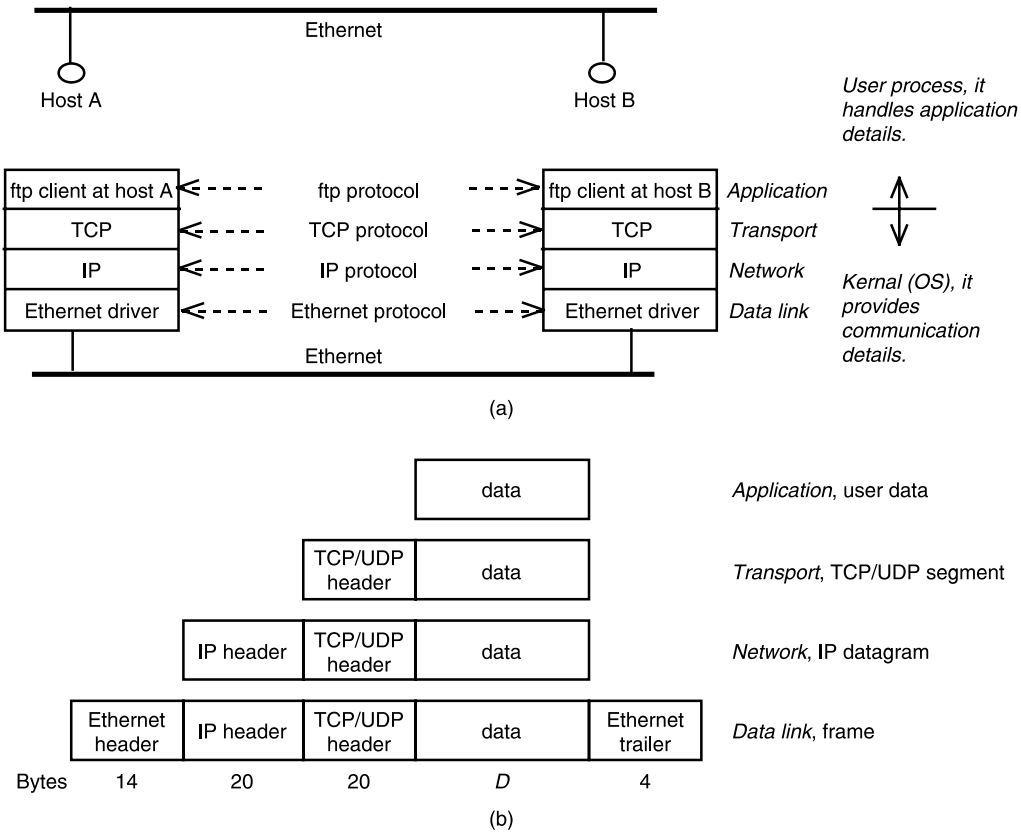


Figure 27.2. Hosts A and B running ftp; (a) a layered model; (b) encapsulation of data as it goes down the protocol stack (note $20 + 20 + D = 46$ to 1500 bytes, supported by the Ethernet).

appending it with an IP header. The data link layer encapsulates the datagram and creates a frame to be handled by the physical layer. The encapsulation requires applying both header and trailer, depending on the type of data link used. At the receiving end, the message flows similarly; but this time the specific layer header is removed and the user data are eventually passed on to the peer application.

The generic term for information combined with an appropriate layer header is Protocol Data Unit (PDU). For example, a TCP segment is a transport layer PDU, and an IP datagram is a network layer PDU.

27.2.3 Internetworking Components

In the following, we consider internetworking components (also called *protocol converters*) such as repeaters, bridges, and routers, and explain their functionality vis-à-vis the OSI model. These components are used to provide an establishment-wide interconnected system. Alternatively, technologies based on FDDI and ATM can also be used for this purpose. Note that two networks may differ at any of the OSI layers. But the design of the protocol converter that establishes a connection becomes more involved as the layer where they differ increases. We also lose forwarding speed as we progress from repeaters through gateways. However, we gain better functionality with this change.

27.2.3.1 Repeater

A repeater that realizes only the physical layer of the OSI model duplicates, amplifies, or regenerates the transmission signal from one network segment onto another network segment. It only interconnects

© 2005 by Chapman & Hall/CRC

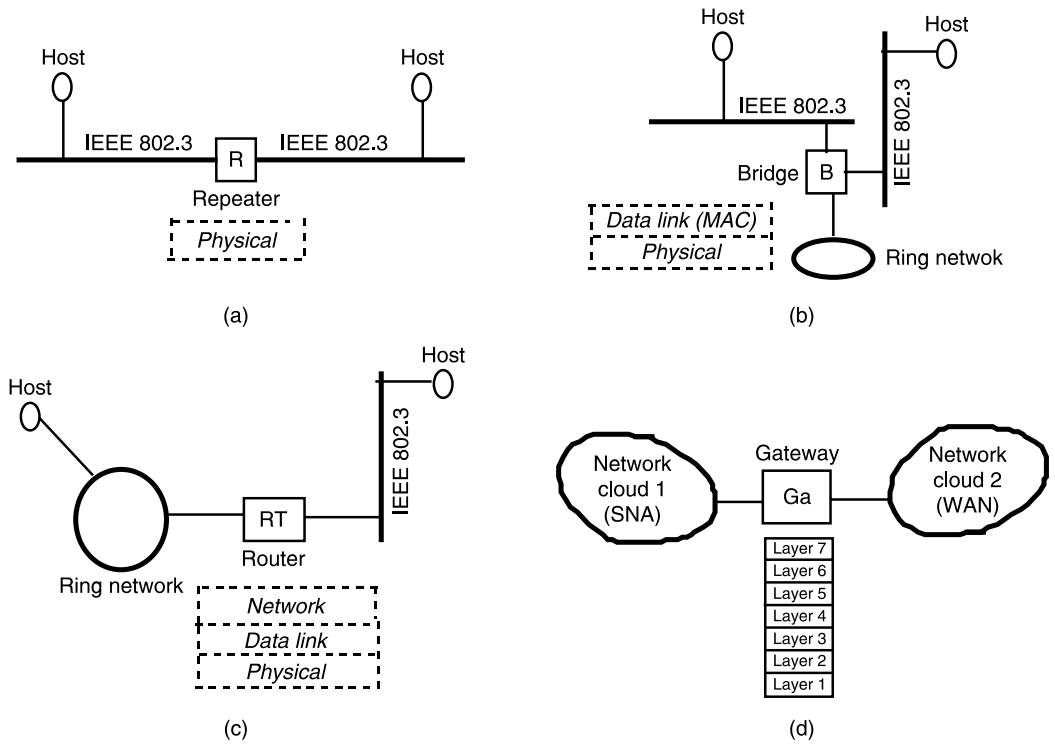


Figure 27.3. Internetworking components: (a) repeater, (b) bridge, (c) router, and (d) gateway (SNA: System Network Architecture; WAN: wide area network).

two homogeneous (identical) networks making a single, larger network [Figure 27.3(a)]. For instance, a repeater helps overcome the segment-length limitation of 500 m on an Ethernet. Since the repeater forwards each bit it receives on the connected segment, the loading or traffic on the entire LAN increases with an increase in demand in the form of number of nodes on the network. This, obviously, causes deterioration in the overall network response time. Also, a LAN has constraints concerning the use of repeaters. Because the repeater takes a finite time to sample a pulse rise and to regenerate the received pulse, it introduces a slight pulse delay — known as *jitter*. As jitter accumulates, it adversely affects the ability of network hosts (stations) to receive data.

27.2.3.2 Bridge

A bridge is a device or layer of software that allows the interconnection of two or more LANs at the media-access (MAC) sublayer of the data link layer of the OSI model. As shown in Figure 27.3(b), different types of LAN are interconnected through a bridge; a LAN segment joins the bridge through a port; and as many segments can be connected as there are numbers of ports in the bridge. A bridge acts in the promiscuous mode, which means it receives and buffers all frames in their entirety on each of its ports. If the frame is error free and the node to which it is addressed is on a different segment, then the bridge forwards the frame onto that segment. Thus, bridges are superior to repeaters because they do not replicate noise or malformed frames; a completely valid frame must be received before it will be relayed. It is important to note that the bridge routing uses the hardware address (of the NIC).

27.2.3.3 Router

As shown in Figure 27.3(c), a router operates at the network layer (or OSI layer 3) and is not sensitive to the details of the data link and physical layers. It contains two or more NICs, just like a bridge. (Any system with multiple interfaces is called MH.) Unlike bridges, an IP address, identifies the interface.

© 2005 by Chapman & Hall/CRC

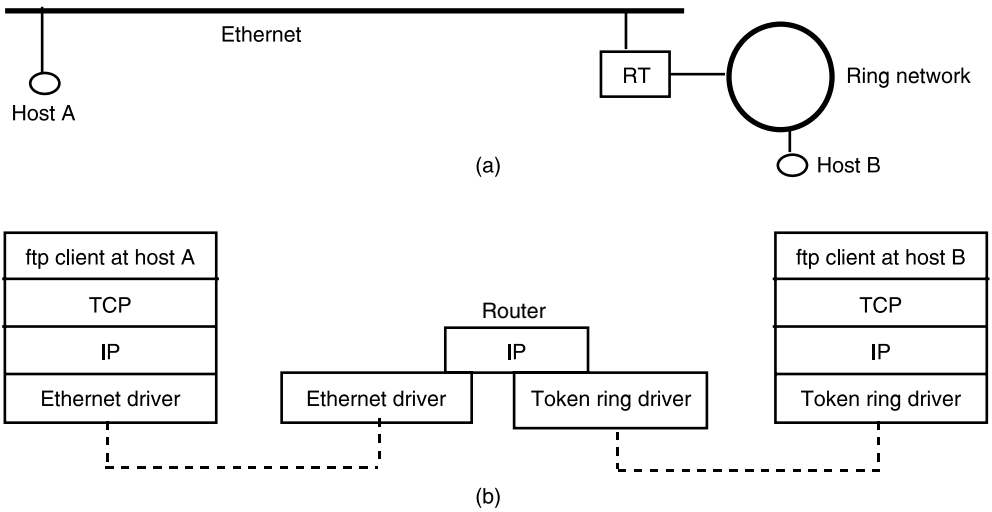


Figure 27.4. Client/server model: (a) a network of Ethernet and token ring; (b) layered representation.

Thus, it can be used to connect different types of network, such as a token ring LAN to an IEEE 802.3 LAN or a LAN to a WAN (see Figure 27.4). A router maintains routing information in routing tables. These tables contain the IP addresses of the hosts and routers on the networks to which the router is connected. The tables also contain pointers to these networks. When a router gets a packet, it consults its routing table to see if it lists the destination address in the header. If the table does not contain the destination address, then the router forwards the packet to a default router listed in its routing table.

Figure 27.3(d), which incorporates all the seven layers, is known as a gateway. Common examples of gateways include Web proxy gateways and transcoders, which convert multimedia data from one format to another. All these example gateways operate on entire messages.

Routers offer an important capability that bridges do not have. This relates to frame segmentation for transmission between LANs. Note that the token ring (Ethernet) supports a data field of up to 4500 (1500) bytes. When a bridge is used to connect these two networks, software must be set at each token ring host to limit all frames to a maximum of 1500 bytes, to accommodate their flow onto a bridged Ethernet network. (Here, the software can be set to limit only those frames destined for an Ethernet host, while still allowing the token ring capability on local hosts.) In contrast, a router can divide a token ring frame into two or more Ethernet frames, eliminating the need of resetting software on each token ring host.

27.3 Link Sharing: Multiplexing and Switching

A high bandwidth[5] (or capacity) line is a costly resource. The cost can be spread among many users by allowing them to share the line by *multiplexing* and *switching*. Multiplexing is, generally, described in the frequency or time domain, where signals of many users are combined into one large bandwidth or long duration respectively. Note that users are geographically distributed and are not confined to a few locations. Switching allows us to bring together the signals of these scattered users by assigning the line on a demand basis for the duration, of time needed, then returning the line back to the pool of available resources after communications have been completed.

[5]We shall use the terms bandwidth (a measure of a link's frequency band in hertz) and capacity (or speed or bit rate in bps) interchangeably. Strictly speaking, these terms differ; can refer to any data communication book for more details.

© 2005 by Chapman & Hall/CRC

On the basis of a frequency or time domain view, a high bandwidth link is considered as a group of frequency or time slots respectively. Let us call it a "channel." In frequency division multiplexing (FDM), each user gets a channel on the high bandwidth link, thus allowing a link to be shared among several users. With time division multiplexing (TDM), a time slot allows the user to utilize the entire bandwidth of the link, but for a very small amount of time (usually of the order of a few milliseconds or less). Contrary to FDM, a signal in TDM is in digital form. Switching, on the other hand, uses switches to allow the link sharing. Different switching techniques are described in this section.

27.3.1 Multiplexing Techniques

Multiplexing contains two Latin words *multi* and *plex*, which mean *many mixing*. It, thus, allows two or more low-speed channels[6] (signals) to share a single high-speed transmission medium such as a wire, cable, optical fiber, etc., of equivalent bandwidth.[7] A multiplexer denotes a device that performs multiplexing according to a frequency or time domain view of a signal. Thus, there are three basic kinds of multiplexing methods: frequency-division multiplexing, time-division multiplexing, and code division multiplexing. They differ from each other in the following ways (refer to Figure 27.5):

1. FDM is an analog methodology allowing users to share frequency slots from the channel's bandwidth. TDM is a digital technique; it means bits and bytes share the time slots on a high-speed line.
2. FDM uses broadband transmission (which means the line bandwidth is divided into multiple channels). It is achieved by translating in frequency or modulating the band-limited signals by separate carriers employing amplitude, frequency, or phase modulation methods, etc. TDM, on the other hand, uses baseband transmission, in which signals are applied to the transmission medium without being translated in frequency.
3. Most FDM units are used to combine very low-speed circuits onto single voice-grade lines for transmission to a central site, while TDMs are used for both low- and high-speed lines for the same purpose.
4. Code-division multiplexing is completely different from FDM and TDM. It allows a station to use all the bandwidth all the time. The basic concept uses a technique called spread-spectrum communication, which was developed to send military signals having antijam capability and covert operation or low-probability of intercept capability. It was also useful for multiple-access, where many users share same band of frequencies. Spread-spectrum communication provides the simultaneous use of a wide frequency band via the code-division–multiple-access (CDMA) technique. In code-division multiplexing, a bit duration is subdivided into n very small intervals: $n = 64$ or 128. The intervals are called *chips*. Each station is assigned an n-bit code or chip sequence. All chip sequences are pairwise orthogonal, such that their normalized inner product is zero. This property is useful, as it helps the decoding process or recovering the correct bit from among the received bits.

27.3.1.1 FDM

As stated above, FDM involves dividing the total bandwidth of a channel into a number of frequency slots, with each slot assigned to a pair of communicating entities. Let the bandwidth of an analog signal $s_i(t)$ be b_i. If the available bandwidth of the high-speed transmission medium is B, then it is divided

[6]A channel is a communication link of fixed bandwidth or capacity.
[7]The term equivalent bandwidth characterizes the source and provides a conservative estimate of its capacity.

© 2005 by Chapman & Hall/CRC

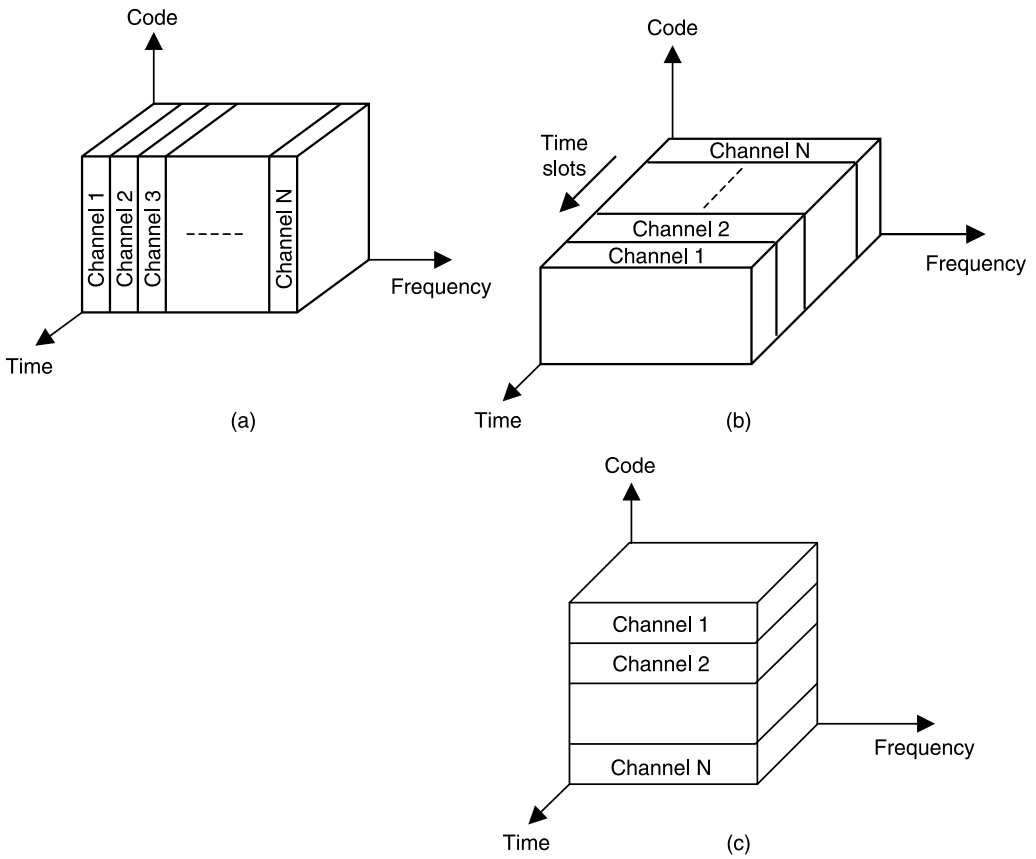


Figure 27.5. (a) Frequency-division, (b) time-division and (c) code-division multiplexing concepts.

into n channels such that

$$B > \sum_{i=1}^{n} b_i$$

The difference in bandwidths, $B - \sum b_i$, is used for guard bands that prevent a channel bleeding into adjoining channels and, thus, help maintain the integrity of the input signals. Figure 27.6 shows some typical aspects of FDM, such as multiplexing (MUX) and demultiplexing (deMUX) frequency translation using modulation, and frequency–time view. Examples of FDM include broadcast radio (both AM and FM) and TV transmissions. For instance, the bandwidth for broadcast TV (54–806 MHz) is partitioned into 68 channels, each having 6 MHz. The very high frequency VHF (ultrahigh frequency UHF) channels 2–13 (14–69) span between 54 and 215 MHz (470 and 806 MHz). FDM offers a specific advantage as it does not need any addressing. The filter and carrier assigned to each frequency slot suffice to separate the signals and direct them to their destination nodes. Note that FDM is effective for analog signals, and is not useful for data communication where we have digital signals. Furthermore, a variation of FDM called *wavelength division multiplexing* (WDM) is used for fiber optic channels.

27.3.1.2 TDM

TDM assigns each signal, preferably in a round-robin fashion, the entire high-speed bandwidth (or capacity, to be very specific) for a time slot (TS), which is a very small time duration of the order of millionths of a second. TDM uses digital signals. Depending on the width of a slot, which could be a bit,

© 2005 by Chapman & Hall/CRC

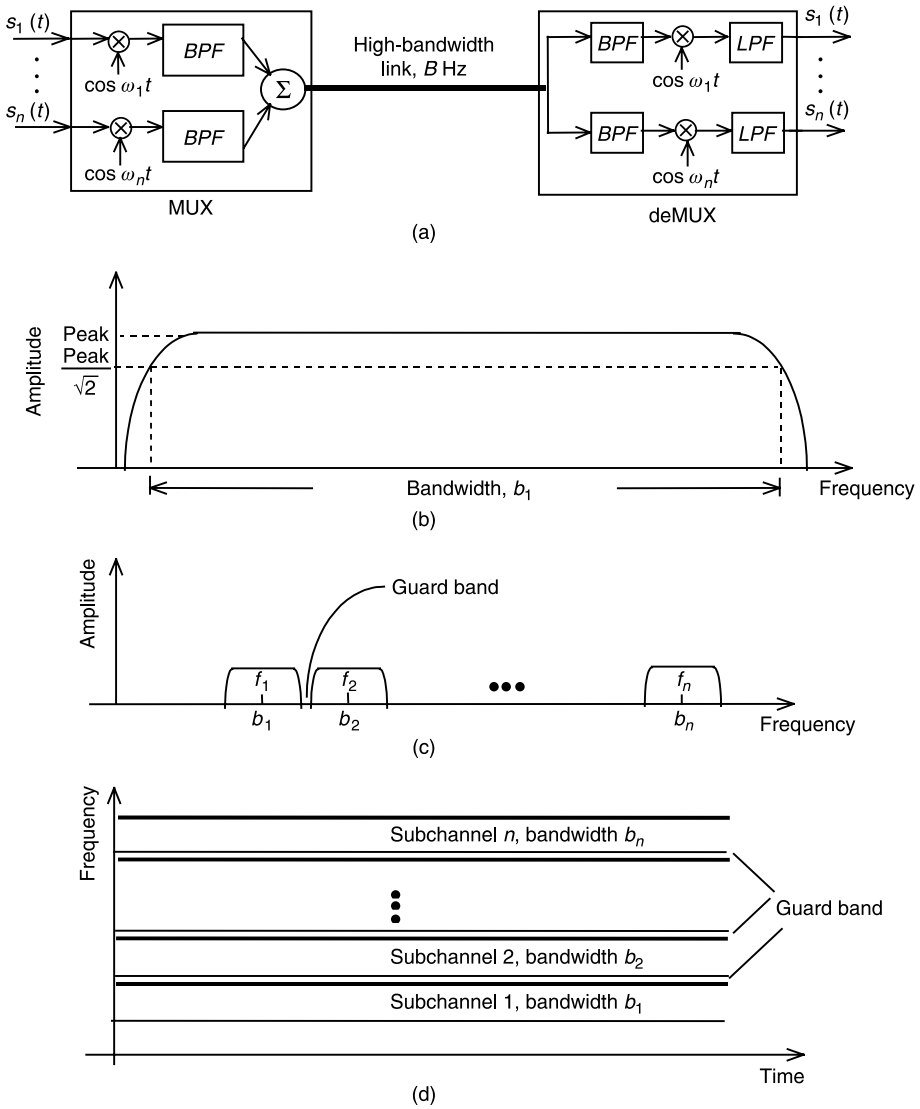


Figure 27.6. FDM: (a) schematic, (b) frequency domain (*BPF*: band-pass filter; *LPE*: low-pass filter; f_i: ith carrier frequency).

byte, or group of bytes, TDMs have three basic modes of operation: bit multiplexing, byte multiplexing, and block multiplexing. A T-1 carrier scheme represents a typical example of byte multiplexing. Others are functionally implemented in exactly the same way. Further, the way a TS is allocated to a channel produces two TDM techniques: *synchronous* and *asynchronous*. Refer to Figure 27.7 for details.

In synchronous TDM, or just TDM, each input has its own preassigned time slot in a frame[8] of size $n \times \tau$, where n is the number of time-multiplexed input signals and τ is the size of a TS. Since the timing of the slot is fixed, it is termed synchronous. If the input has no data to send at that moment when its turn comes, then the TS remains empty because no one else can use it. This is also the case with FDM. In both cases, the capacity of a costly resource is wasted. Figure 27.7 shows a synchronous TDM scheme.

[8]A frame consists of a sequence of slots: slot 1, slot 2, . . . , slot n. A logical channel employs every nth slot. Thus, a logical channel i occupies slots i, $i+n$, $i+2n$, . . . , and so on.

© 2005 by Chapman & Hall/CRC

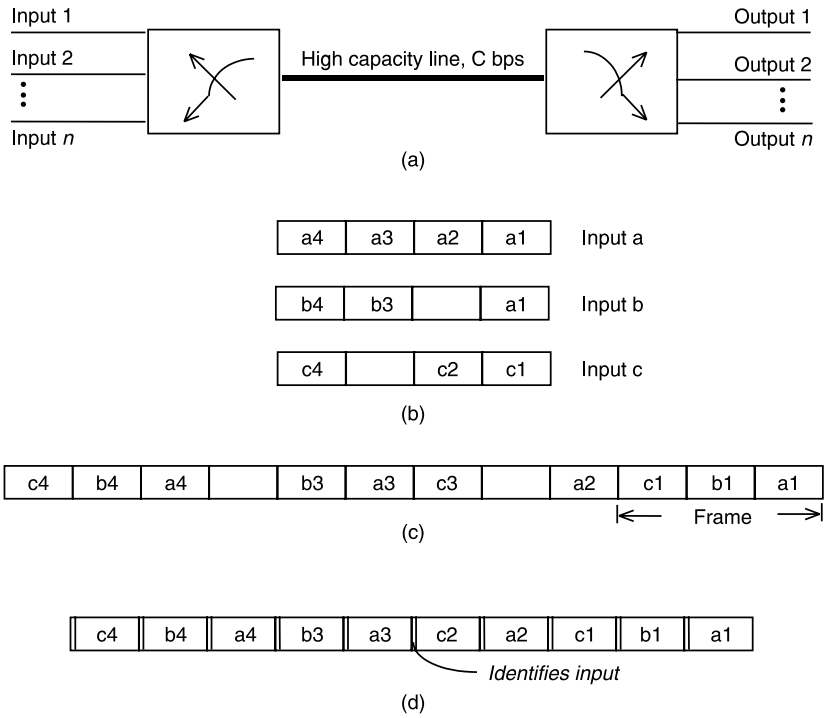


Figure 27.7. TDM: (a) schematic; (b) slots from three inputs a, b, and c; (c) synchronous TDM signal; (d) asynchronous TDM signal on high-capacity line.

It is obvious from this illustration that if c_i is the input rate of the ith source and C is the capacity of the aggregated link, then $C \geq \sum c_i$. Synchronous TDM is good for delay-sensitive traffic, such as voice and broadcast TV signals. If used with data communication that is generally bursty, then the link utilization[9] will be less because there exist longer quiet periods (when there is no data on the line) with burst traffic.

In an asynchronous (or statistical) TDM, a TS can be used by any input as long as it is not occupied, and hence the term asynchronous. This is achieved by assigning an identifier for the particular input in each time slot. Figure 27.7(d) shows an asynchronous TDM scheme. It is obvious from this illustration that if c_i is the input rate of the ith source and C is the capacity of the aggregated link, then $C < \sum c_i$. Thus the ratio of $\sum c_i/C$, called *multiplexing gain*, is greater than one (note that it is at most one for synchronous TDM). Alternatively, it states that more than n inputs can be accommodated over the channel having a capacity of C bps. This, though, imposes a performance restriction in terms of *bit error rate* (BER) tolerance. Also, buffering is needed to avoid overflows, especially when several incoming traffic bursts coincide. Asynchronous TDM is useful for delay-insensitive data traffic, because it helps improve link utilization by dynamically assigning time slot duration to active users.

27.3.2 Switching Techniques

Recall that switching helps apportion a line/channel[10] among users on a demand basis. À la carte, if available, a channel is assigned to a user for the duration of time needed. Once the duration is over,

[9]The link or channel *utilization* refers to the average data rate divided by channel capacity or the fraction of time a channel is busy.

[10]We assume that the capacity of a channel is partitioned into a number of fixed-rate *circuits* or *logical channels*. The apportionment is usually achieved using TDM. For example, T-1 contains 24 voice or data circuits.

© 2005 by Chapman & Hall/CRC

the channel is returned back into a common resource pool where it is accessible to other users. A switch[11] or a router performs the switching function, which includes: (a) access control (i.e. whether or not to accept the call from the requesting user), (b) determining the appropriate route, and (c) resource allocation (means assigning a bandwidth or capacity) along the route to be used to transfer the data stream. Switching techniques, described below, can be distinguished based on these three features. Another point of differentiation comes from the concept of channel utilization, defined as the ratio of busy time to the total duration or fraction of time the channel is busy. Using this, the transmission of a voice signal has a utilization of around 50%. A data communication, generally termed *bursty*, can have utilization that ranges from 0 to 100%. It is very low in the case of interactive data (telnet and rlogin, for example) and high in the case of bulk data (ftp, e-mail, Usenet news). Besides utilization, other parameters that describe a call include origin and destination, bandwidth (peak, average, or effective), delay constraint, and admission error rate. Considering the scope of the book, we shall exclude these other parameters from our discussion below. We now describe the three most important types of switching technique: circuit switching, message switching, and packet switching.

27.3.2.1 Circuit Switching

Circuit switching, a method popularized by telephony, requires three steps. Step 1 is a *call* or *connection* set-up phase. Here, a source attempts to establish an end-to-end physical connection by generating a call request signal to the nearest switch of the network to which it is connected. As the request hops from node to node towards its destination, a circuit from each link along the route, if available, is reserved by the network. The circuit may contain a frequency band or a slot depending on FDM or TDM being used. The route decision utilizes a simple strategy, such as the first path available from a fixed list, or a complex measure, like taking a path with the largest residual capacity. If the circuit assignment on links en route to the destination is successful, then the network is said to *admit the call*. However, if it fails on any one of the links, then the call is said to be blocked or rejected (depending on the fact that it is made to wait until the resource becomes available, or is cleared). The network is then termed *busy*. Step 2 is called the *call holding* phase and performs data transfer once the connection is established. Thus, an admitted call has a fixed route and bandwidth during the data transfer phase. A particular frequency band or a TS assigned in step 1 is not available to anyone else as long as the data are transmitted. This makes flow control within the call unnecessary. Further, during this phase, the data stream does not undergo any buffering at intermediate nodes. This eliminates queuing and processing delays at these switches. There exists propagation delay, however, which is about 0.5 ms for every 100 km line segment. Circuit switching is ideally suited for audio and video transmissions, which represent delay-sensitive signals. Step 3 is known as the *call clearing* phase and is performed when the data transfer is complete. In this step, the connection is deallocated, which means the circuit is returned back to the bank of available resources for future assignments. Figure 27.8 provides an example of a circuit switched network where (s,t) refers to a source-destination node pair.

Circuit switching is not efficient for data traffic. As mentioned earlier, a data stream is bursty. This means that transmissions are short and occur irregularly. In other words, periods of peak transmission are followed by intervals when there are no data on the line. Note that a circuit switching scheme reserves the line/channel for the entire duration of the transfer; thus, a user is billed for the data transfer periods in which the line remains used. The problem becomes acute with interactive data transfer. Another reason that makes circuit switching unattractive for data traffic is the varying rate of data transmission. Note, data rates differ widely — from hundreds of bps between a terminal and the computer to millions of bps between two computers. A system that should provide circuit-switched connectivity to varying rates needs to be designed keeping maximum rate in view. This is because with peak demand the system will fail to work, as circuit-switched data cannot be delayed. Designing the entire system to take care of this situation, evidently, presents a waste on resources.

[11]The term *switch* is used in telephony, whereas a *router* is employed in computer communication. We shall use switch and router in the same sense.

© 2005 by Chapman & Hall/CRC

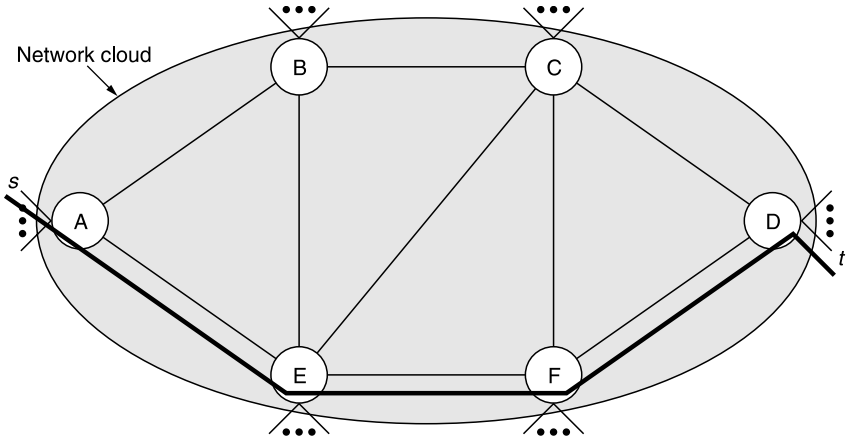


Figure 27.8. Illustrating circuit switching.

27.3.2.2 Message Switching

CYBERNET, implemented by Control Data Corporation (CDC) using switched, leased, and private lines and satellite links, provides an example of a message switched network. A message is a variable-length bit string and characterizes a logical unit of information for the purpose of communicating to one or more destinations. Programs and data files present two typical examples of messages. Similar to processing telegrams in the telegraph industry, message switching is based on a store and forward concept. This means that message switching does not require establishing an *a priori* physical connection between a source–destination pair. Here, the sender prefixes the destination address to a block of data and sends it to its neighbor. The neighbor stores the data and later forwards it to its neighbor en route to the destination. The message is thus transmitted in its entirety and hop by hop through the network until it finally reaches the destination. This scheme provides efficient channel utilization. Since messages wait in a queue before being transmitted and they also arrive at a node randomly depending on the congestion in the network, a message in this switching technique encounters queuing and transmission delays in addition to propagation delay. Further, no restriction is imposed on message lengths, which can differ vastly. This implies an intermediate node storing the information before forwarding to its neighbor should have enough disk space for the varying message sizes. To get around this problem, the concept of packet switching is used.

27.3.2.3 Packet Switching

Similar to message switching, a packet is stored and then forwarded by successive nodes en route to the destination node in the packet-switched network. A *packet* is a small fixed-size block of the message. Each packet contains a sequence number and a label specifying the destination address, and is individually transmitted. Such fragmentation of a message offers the following advantages. Because the size of a packet is known *a priori*, its storage at intermediate nodes during transmission between source and destination is manageable. Also, a packet i that has been received and processed at some node j can be forwarded before packet $(i+1)$ fully arrives at j. This obviously reduces delays and improves the throughput. Figure 27.9 illustrates the working of a packet-switched network. We assume that a message containing five packets, labeled 1 through 5, traverses between switches A and D. These packets are generated at source s and are destined for destination t. Node A transmits alternately to nodes B and E, while intermediate switches decide the next hop depending on a routing strategy in accordance with traffic congestion, the shortest end-to-end path, and other criteria.

Let us now consider functions such as access control, routing, and resource allocation to help compare packet and circuit switching methods. The process of call acceptance and routing for packet-switched calls is similar to the corresponding processes for circuit-switched calls. For example, a switch

© 2005 by Chapman & Hall/CRC

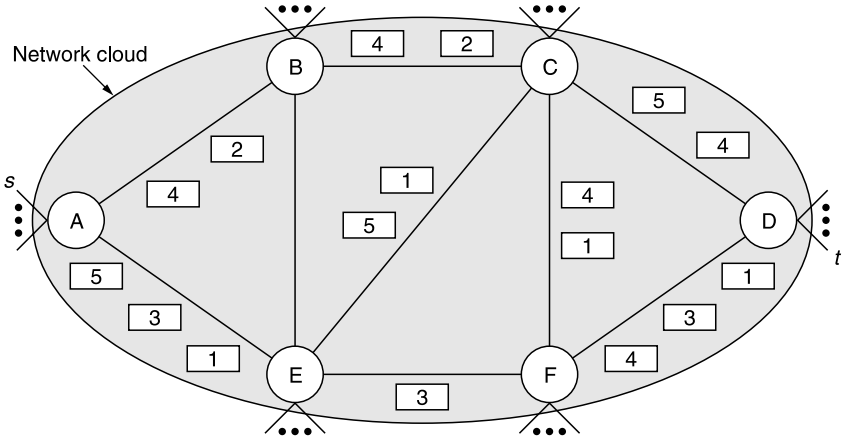


Figure 27.9. Packet switching concept and path identifiers.

in packet switching may generally accept all arriving packets or may use some discretion based on parameters such as cost, delay, throughput, reliability, etc. A routing protocol then employs these measures to determine a source to destination path. Typically, the call is connected on the paths with the minimum expected end-to-end delay[12] within a given list of available paths. The capacity allocation problem provides an important point of difference between packet and circuit switching techniques. We make a network-wide resource allocation in circuit switching, and that, too, for the complete duration of the call. Thus, there exists a real or deterministic commitment of resources. In packet switching, an allocation is done for individual packets for the duration of the transmission from one switch to the next as they progress within the network. The commitment is therefore virtual or stochastic. To illustrate this point, if the capacity allocation is made on an average bandwidth basis and bursts occur at peak rates, then the network, depending on the policy that is currently being enforced, may drop all or part of the data packet stream. Also, this necessitates the use of a flow control scheme, mostly on a link basis using on–off or continuous (selective or go-back to *n* type) ARQ scheme.

We have already mentioned that a switch decides a bandwidth or capacity and an appropriate path to the requesting user. In packet-switched networks, packets from different sources that share a common link are statistically multiplexed. This helps optimize the bandwidth allocation by providing a multiplexing gain, as discussed in Section 27.3.1.2.

The selection of a route that packets should follow is determined in one of two ways: virtual circuit and datagram. Figure 27.10 illustrates these services. The virtual circuit (VC) or connection-oriented transport service is similar to circuit switching. In this case, a path called a virtual circuit is set up through the network for each session [Figure 27.10(a)]. Switches along the route then contain a mapping between incoming and outgoing VCs. Different packets that are part of the same data transfer use this information for their routing decision. Thus, they follow the same path and packets arrive in order at the destination, eliminating the need of a packet assembler/disassembler (PAD). At the end of the session, like circuit switching, the VC is cleared. A VC service is further classified as switched or permanent. The switched VC (SVC) is similar to a dial-up or public connection, while a permanent VC

[12]Typical acceptable delay depends on the type of data stream. For real-time applications, it is about 200 ms; for interactive traffic (noninteractive services such as e-mail), it is a few (many) seconds.

© 2005 by Chapman & Hall/CRC

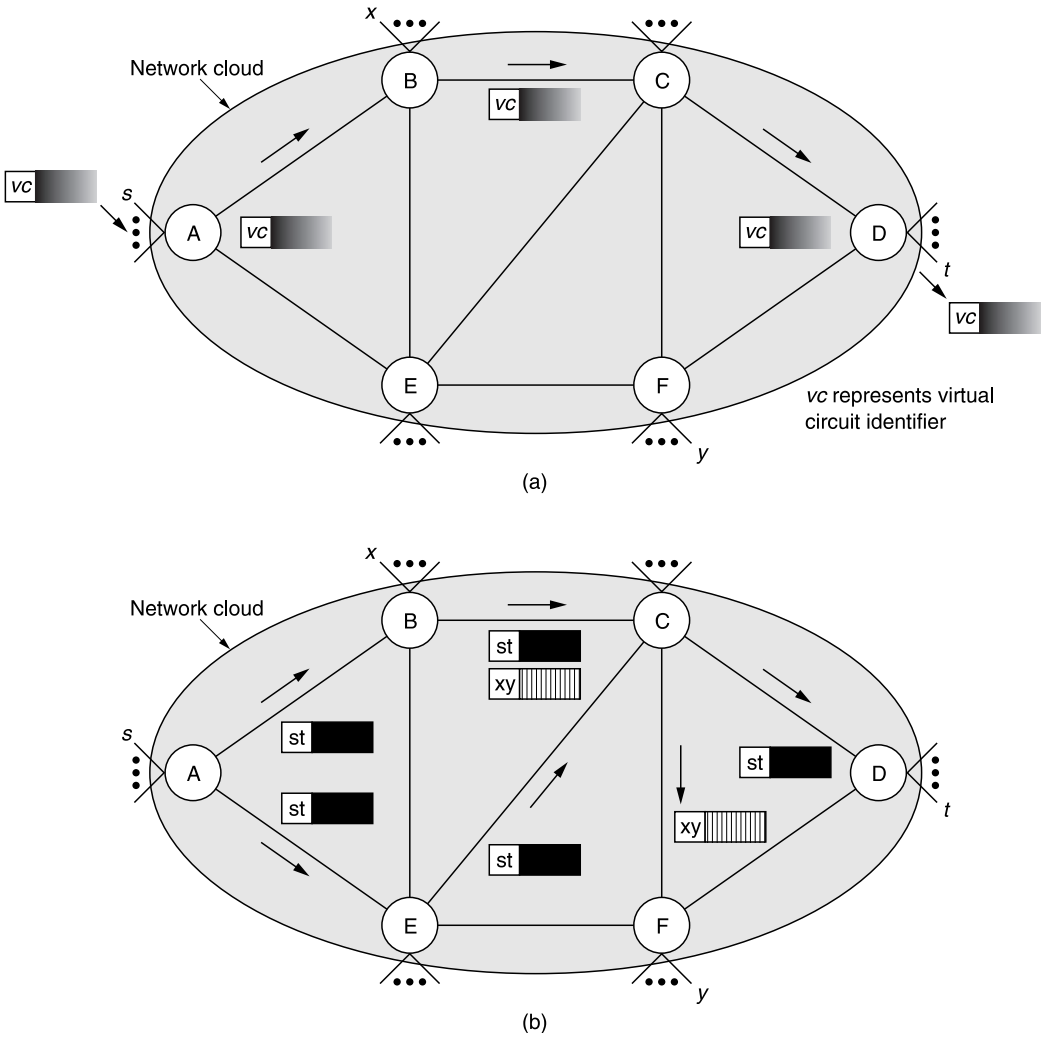


Figure 27.10. Two packet-switching techniques: (a) virtual circuit; (b) datagram.

(PVC) is analogous to a leased or private line. Thus, a PVC does not need a call set-up phase. There is, though, a subtle difference between a private line and a PVC connection. In PVC, the bandwidth of the line is shared among multiple users, while the entire bandwidth of the line is dedicated when it is leased. TCP provides a typical example of a VC service.

A datagram (or connectionless) transport service, on the other hand, is very similar to the way letters are handled by the post office. Each letter, marked with its destination address, goes from one post office to another post office, and so on, until it reaches its destination. In datagram switching, a packet is labeled with its source and destination addresses and is transmitted individually [see Figure 27.10(b)] (for *s–t* and *x–y* traffic). The inclusion of address bits (which are often quite long) with each packet provides an overhead. The routing decision for each packet is made at every intermediate switch using a static or dynamic (meaning periodically updated) routing table. Note: the table is created based on some optimality criterion, such as cost, delay, distance, throughput, etc. It is likely that a routing table may change because of changing conditions when a node or link fails or is unavailable because of congestion. Datagram packets react quickly to the changes, thereby meaning that packets from the same source may follow different paths to arrive at the destination and also that their arrival could be out of order. Hence, a destination node should be capable of resequencing the packets; an expensive PAD

© 2005 by Chapman & Hall/CRC

handles this job. The datagram transport service does not need any prior path set up; hence, it is quite suitable for short transmissions of a few packets. UDP provides an illustrative example.

Example 27.2. To illustrate the computation of delays in VC and datagram services, consider a typical route (AB, BC, CD) in Figure 27.10 taken by a packet having 500 bits of data in either of the services. In addition to data bits, we assume in the VC case a header of 5 bytes for VC number and a trailer of 2 bytes to support error correction. Further, let the VC set-up time be 250 ms. Thus, the transmission time in VC is $(500 + 8 \times 5 + 8 \times 2)/56{,}000$ which is 9.93 ms. Here, we assume that the capacity of each link is 56 Kbps. To transmit N packets, we require a total time $T(\text{VC}) = 250 + (N + 3) \times 9.93$ ms because three links, AB, BC, and CD, are involved. To compute these parameters for a datagram service, consider the header part as 10 bytes instead of 5 bytes, as it indicates both source and destination addresses. The trailer part, however, remains unchanged as 2 bytes. Thus, the transmission time in a datagram is $(500 + 8 \times 10 + 8 \times 2)/56{,}000$, which is 10.64 ms and $T(\text{datagram}) = (N + 3) \times 10.64$ ms.

Example 27.3. Obtain N for which $T(\text{VC}) < T(\text{datagram})$ in Example 27.2.

27.4 Data Transmission Basics

Several parameters, such as medium, bit and character encoding schemes, half- or full- duplex, serial or parallel transfer, and asynchronous or synchronous type, help characterize a data communication arrangement such as PPP. A twisted wire, cable, and optical fiber form some typical examples of the communication medium. Unshielded twisted wire[13] pair (UTP) is mainly utilized in telephone networks. A LAN based on 10BaseT[14] also uses UTP. A shielded twisted wire pair, cable, or optical fiber provides better bandwidth. For example, 10Base2 employs a thin coaxial cable, whereas 10Base5 utilizes a thick coaxial cable. The distance Limit for thin (thick) cable is 185 (500 m) per segment. Both 10Base2 and 10Base5 are used in LANs. Line codes, such as nonreturn to zero (NRZ), Manchester, alternate mark inversion (AMI), B8ZS, 4B/5B etc., characterize typical bit encoding schemes. ASCII and EBCDIC codes represent the two most popular character encoding schemes in data communication. They provide unique 7 or 8 bit patterns in order to identify all the characters on the keyboard. A simplex mode refers to data communication in only one direction. On the contrary, the capability of data transmission alternatively or simultaneously in both directions is designated as half- or full-duplex mode respectively. In the following, we describe the fundamental concepts associated with the bit-serial transmission because of its ubiquitous usage in computer networks.

27.4.1 Serial and Parallel Modes

There are two ways to transmit the bits that make up a data character: serial or parallel. Serial transmission sends bits sequentially, one after the other, over a single wire (or channel). It is further classified as asynchronous or synchronous type. In parallel transmission, bits of a data character are transmitted simultaneously over a number of channels, which are generally organized as a multiple of eight. Figure 27.11 provides an illustration for bit-serial and bit-parallel transmission modes. Although serial transmission is slower than parallel, it is predominantly used between host and computer today. A typical example of parallel transmission is found with address, data, and control buses between a

[13]A twisted wire pair is better than a two-wire medium because any noise interference affects both wires (not just one) in the pair and hence gets canceled. In addition, the effects of crosstalk can be minimized by enclosing a number of twisted pairs within the same cable.

[14]Each IEEE 802.3 physical layer has a name that summarizes three characteristics of a LAN. For example, the first parameter 10 in 10Base5 refers to the speed of the LAN in Mbps. The word 'Base' identifies a baseband technique; 'Broad' denotes broadband method. Finally, the 5 provides the LAN segment length, in multiples of 100 m. Here, 'T' means a twisted wire pair.

© 2005 by Chapman & Hall/CRC

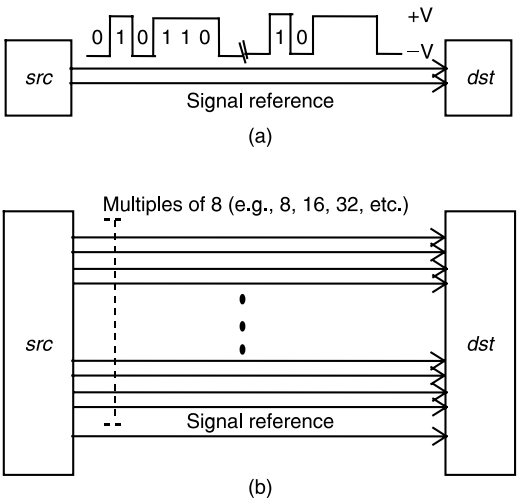


Figure 27.11. Illustrating (a) bit-serial and (b) bit-parallel transmission modes (*src* denotes source, *dst* means destination).

microprocessor and external devices such as memory and input/output modules. Considering the scope of the chapter, we will not elaborate further on parallel transmission modes, such as the High Performance Parallel Interface (HIPPI), which is an American National Standard Institute (ANSI) standard and is adopted by supercomputer, workstation, and peripheral manufacturers as the high-performance interface of choice.

27.4.2 Transmission Type

As we mentioned at the beginning of this chapter, a frame is the logical group of information sent as a data link layer unit over a transmission medium. The frame thus contains bytes, and bytes embody bits (Figure 27.12). For the receiver to interpret the bit pattern correctly, the bit (clock) period, start and end of byte (character), and start and end of frame must be uniquely resolved. These tasks are called *bit* (clock) *synchronization*, *byte* (character) *synchronization*, and *frame synchronization* respectively. They are accomplished in one of two ways, depending on bit-serial transmission being asynchronous or synchronous type.

The asynchronous mode treats each character (byte) independently, and hence embeds them with additional control bits [refer to Figure 27.12(a)]. This makes the approach useful when the data to be transmitted are of low volume or generated randomly. The asynchronous mode of communication is normally used for low data rates, up to 19.2 Kbps.[15] A typical example of data being generated randomly is a user sending data from the keyboard to the computer.

Synchronous transmission, on the contrary, permits the transmission of a complete frame as a contiguous string of bits, and hence it does not use the control bits on a per-character basis [see Figure 27.12(b)]. It is primarily used in a large-volume data transfer, as could happen if a file is being transferred between two computers. Synchronous transmission is also an alternative to transmit data at higher bit rates.

27.4.2.1 Asynchronous Transmission

We have just indicated that each character in an asynchronous mode is embedded with control bits called start, stop, and, if applicable, parity bits [see Figure 27.12(a)]. One start and one, one and a half,

[15]Note that EIA-232-D interface specifications limit the data rate to 19.2 Kbps with a 50 ft. cable. But, due to improved techniques, higher data rates such as 28.8, 33.6, and 56 Kbps are available these days.

© 2005 by Chapman & Hall/CRC

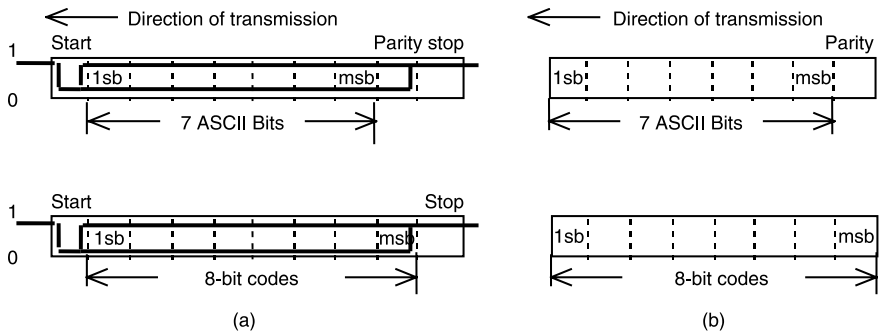


Figure 27.12. Byte structure in a bit-serial (a) asynchronous and (b) synchronous transmission.

or two stop bits designate the beginning and ending of bytes. The presence of an even, odd, MARK, or SPACE parity[16] bit helps detect the transmission errors at the byte level. The accepted mode of asynchronous transmission is to hold the serial line at a logic 1 level until data are to be transmitted. The transmission begins with a logic zero for one bit time (start bit). The principle of bit and character synchronization is as follows: the receiver contains a circuit having a serial-in parallel-out shift register, a local clock that is N times faster than the transmitted bit rate (usually $N = 16$), and a divide by N counter. The local clock is chosen to be faster because the receiver clock runs asynchronously with respect to the transmitter clock. After detecting a $1 \rightarrow 0$ transition associated with the start bit of a byte, the first bit is sampled at $N/2$ clock cycles and then subsequently after N cycles for each bit in the character. In this way, we ensure that each bit is sampled approximately at the center of the bit cell. This procedure is repeated at the start of each character received. The sampled bits, retained with the serial-in parallel-out shift register, define a character byte. The frame is encapsulated using reserved (transmission control) characters such as the END byte in SLIP (serial line IP) and MARK in Kermit. Since the frame is transmitted character by character, its synchronization is automatically achieved when bits and bytes are synchronized. The need for additional bits at both character and frame levels makes the asynchronous mode inefficient in its use of the line capacity. For example if we assume that 100 data characters are to be transmitted using SLIP, then the minimum transmission overheads at the byte level add up to 200 bits (with one start bit and one stop bit per 8-bit character) and at frame level to 16 bits (considering that no byte stuffing is required). This means the transmission efficiency, defined as the ratio of data bits to the total number of bits actually transmitted, is 800/1016 or 78.7%. This will further decrease if the inclusion of parity bits and character stuffing is assumed.

Example 27.4. Let R bps be the data rate in an asynchronous transmission. If CK denotes the receiver clock frequency, then bit synchronization requires $\text{CK} = N \times R$. This means the bit period is N/CK. The mid-point sampling of the subsequent bit cells depends on the detection of the $1 \rightarrow 0$ transition associated with the start bit in each byte. Thus, a detection error will cause sampling to deviate from the center of the bit cell. It is shown that the worst-case deviation D from the nominal bit cell center is approximately one cycle of the receiver clock (1/CK). Alternatively, $D = (1/N) \times$ bit period. Consider CK = 19.2 kHz and transmission data rate R in bps is (i) 2,400, (ii) 9,800, and (iii) 19,200. The parameter D, in terms of corresponding bit period, is (i) 1/8, (ii) 1/2, and (iii) 1 respectively. The chance of missing a bit is 50% in case (ii) and 100% in case (iii). The corresponding data rates are unacceptable with CK being 19.2 kHz.

27.4.2.2 Synchronous Transmission

From Example 27.4, it is evident that the bit synchronization in an asynchronous mode becomes unreliable with an increase in data rate. Synchronous transmission is used to overcome this problem.

[16]The parity is termed even if the modulo-2 sum of all the bits, including the parity bit, is even. For an *odd parity*, the sum is odd. MARK parity, regardless of data bits, is always one. The SPACE parity is similarly always zero.

© 2005 by Chapman & Hall/CRC

As its name suggests, synchronous mode is the operation of a network system wherein events at both transmitter and receiver occur with precise clocking. There are two types of synchronous transmission, namely byte-oriented and bit-oriented. Both use the same bit synchronization methods that could be based on a circuit having a digital phase-lock loop which exploits $1 \rightarrow 0$ and $0 \rightarrow 1$ transitions present in the received bit stream. The byte and frame synchronization are, however, handled differently in the two systems. In byte-oriented transmission, the transmitter sends two or more ASCII SYN characters preceding each frame. Its bipolar coding, on the one hand, helps maintain synchronization of transmitter and receiver clocks. On the other hand, the SYN character itself provides the correct byte boundaries. With bit-oriented transmission, the transmitter continuously sends idle bytes (0111 1111) on the line during the period between the transmission of successive frames. Here also, a bipolar encoding of the idle bytes either using the Manchester scheme or its variants provides clock alignment. Note, in both types of synchronous transmission, the characters are not embedded with control bits because the frame is transmitted as a bit stream [Figure 27.12(b)]. Nonetheless, similar to asynchronous mode, a reserved transmission control byte or character is used for frame encapsulation in both types of synchronous transmission. With byte-oriented, it is generally ASCII start-of-text (STX) and end-of-text (ETX) codes, while it is a flag pattern 0111 1110 in the case of bit-oriented type. IBM's Binary Synchronous (BiSync) protocol provides an example of a byte-oriented protocol. Considering the scope of the book, we have excluded its description. A typical example of bit-oriented transmission is PPP or high-level data link control (HDLC). Both byte-oriented and bit-oriented types of synchronous protocol contain additional fields that make them flexible and help support a variety of link types and topologies. For instance, HDLC can be used with point-to-point and multipoint links, half- and full-duplex transmission facilities, and circuit-switched and packet-switched networks. Further, with the help of cyclic redundancy check (CRC) and ARQ schemes, the synchronous transmission scheme achieves link error recovery.

27.5 Wireless Networks

Flexibility and mobility make wireless networks one of the fastest growing areas in the telecommunications industry. Wireless networks link users and information services through a wireless communication path or channel. They offer both LAN and WAN connectivity for business and home users. Different types of wireless network have been used for the different types of service. Satellite systems provide high-speed broadcast services, and low-speed long-distance (even international) digital voice services. Cellular systems provide radio coverage to a wide area, such as a city. A sensor network is formed when a set of small sensor devices that are deployed in an *ad hoc* fashion cooperate on sensing a physical phenomenon. Sensor networks greatly extend the ability to monitor and control the physical environment from remote locations, e.g. it can be used to analyze the motion of a tornado, detect fires in a forest, etc. Wireless LAN (WLAN) is designed to operate in a small area, such as a building or office, and allows computers and workstations to communicate with each other using radio signals to transfer high-speed digital information. This section emphasize some principles and technologies of wireless networks, based on cellular systems.

27.5.1 Terminology and Model

Various wireless communication systems, such as remote controllers, cordless telephones, pagers, walkie-talkies, cellular telephones, and so on, are currently being used. Their mobility allows users to move during operation. Unlike wired links, which usually provide one-to-one communication without interference, wireless links use one-to-many communication where some problems exist, such as noise, interference, bandwidth limitations and security, etc. Mobile radio transmission systems may be classified as simplex, half-duplex and full-duplex. Simplex systems provide only one-way communication, such as paging systems. Half-duplex systems allow two-way communication, but use the same

© 2005 by Chapman & Hall/CRC

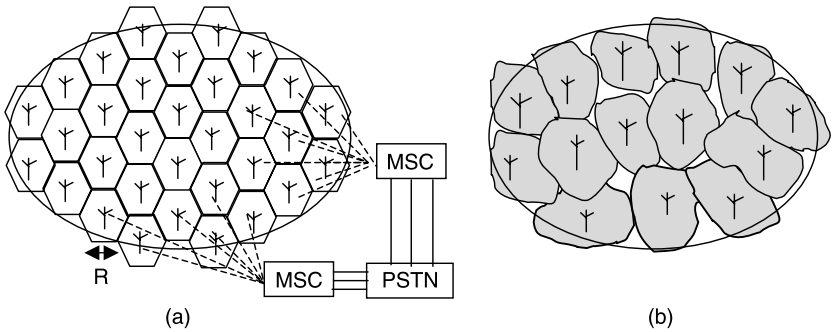


Figure 27.13. (a) A typical wireless network model (cellular system) with uniform hexagonal-shaped areas called cells, MSC. (b) An illustration of coverage areas in a real-world wireless network system.

radio channel for both transmission and reception, such as ''push-to-talk'' and ''release-to-listen'' systems. At any time, users can only either transmit or receive information. Full-duplex systems, on the other hand, allow simultaneous radio transmission and reception by using two separate channels. All wireless networks that we consider in this section are full-duplex systems.

The mobile stations communicate to fixed base stations (BS), which are located at the center or on the edge of a coverage region and consist of radio channels and transmitter and receiver antennas on a tower. A Mobile switching center (MSC) coordinates the activities of all of BS in a large service area. The service area is the total region over which the BS are distributed. Coverage area is the actual region over which communications can be provided. The ratio of coverage area to service area is called *area availability*. Radio channels used for transmission of information from the BS to the mobile station are called forward channels (downlinks); radio channels used for transmission of information from the mobile station to the BS are called reverse channels (uplinks). Some specific radio channels are assigned as control channels, which are used for transmission of call setup, call request and other beacon or control purposes. Similarly, control channels include forward control channels and reverse control channels. System capacity is defined as the largest number of users that could be handled by a system. Figure 27.13 shows a very common wireless network model.

27.5.2 Frequency Reuse and Channel Assignment

A frequency spectrum allocated to wireless communication is limited. By replacing a single, high-power transmitter (covering a large service area) with many low-power transmitters (a large service area being divided into many small cells), the cellular networks reuse the frequency at a smaller distance such that system capacity is increased.

Frequency reuse is possible due to the propagation properties of radio waves. The transmitted power is chosen to be just large enough to help communication with mobile units located near the edges of the cell. Each cell is assigned a certain number of channels. To minimize the interference between base stations, the channels assigned to one cell must be completely different from the channels assigned to its neighboring cells. By carefully spacing base stations, properly setting the distance D and assigning their channel groups, the same frequency can be used in the two (or more) cells simultaneously as long as the interference between co-channel stations is kept below acceptable levels. Interference coming from a reused frequency is called co-channel interference. The distance D is called the reuse distance. Figure 27.14 illustrates the concept of frequency reuse. Cells with the same number use the same group of channels (frequencies). A cell cluster is outlined in bold and replicated over the coverage area.

Maximizing the number of times each channel may be reused and minimizing interference at the same time in a given geographic area is the key to an efficient cellular system design. Co-channel

© 2005 by Chapman & Hall/CRC

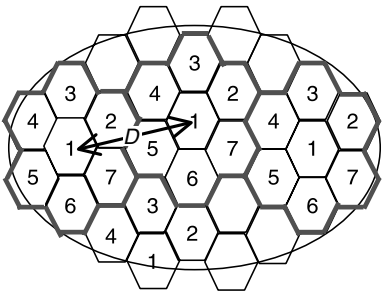


Figure 27.14. Illustrating frequency reuse.

interference ratio at the desired mobile receiver depends on cell radius R and the reuse distance D.

$$\frac{S}{I} = \frac{S}{\sum_{k=1}^{N_I} I_k} = \frac{1}{\sum_{k=1}^{N_I} \left(\frac{D_k}{R}\right)^{-\gamma}}$$

where $S(I)$ is signal (interference) power and I_k and D_k refer to the kth cell.

Here, γ is the propagation path-loss slope, which depends on the terrain environment; N_I is the number of co-channel interfering cells, e.g. in a fully equipped hexagonal-shaped cellular system, $N_I = 6$ for the first tier.

Channel assignment strategies are classified into two groups: fixed or dynamic. In a fixed channel assignment (FCA), each cell is assigned a predetermined set of channels. FCA is simple and works well when the traffic follows a uniform distribution. However, it behaves very poorly in the worst case. In a dynamic channel assignment, channels are not assigned to cells permanently. All channels are kept in a central pool and are assigned dynamically to cells as new calls arrive in the system. A channel is eligible for use in any cell as long as interference constraints are met.

27.5.3 Handoff

The handoff process mainly consists of four steps: initiation, resource reservation, execution, and completion. It is used to switch channels automatically to maintain a conversation in progress as the mobile terminal moves into a different cell. Handoff is very important in any cellular radio system and must be performed successfully as infrequently as possible, and be imperceptible to the users. The operation not only involves identifying a new BS, but also requires that the voice and control signals be allocated to channels associated with the new BS.

Handoffs can be classified as soft handoff or hard handoff. In a soft handoff, the mobile terminal can communicate with two radio ports simultaneously, such that the commutation will not be interrupted when the mobile terminal moves into a different cell. A hard handoff occurs when the old connection is broken before a new connection is activated, due to disjoint radio systems, different frequency assignments, or different air interface features. It is a ''break-before-make'' process at the air interface.

27.5.4 Multiple Access Technologies

Section 27.3.1 discussed FDM and TDM systems. Applying these principles, we have frequency division multiple access (FDMA) and time division multiple access (TDMA) technologies to handle multiple access where many users share a finite amount of radio spectrum simultaneously.

Frequency Division Multiple Access (FDMA) systems divide bandwidth of the air interface between the mobile station and the BS into multiple equal analog channels, each of which occupies one part of a larger frequency spectrum. Then, FDMA assigns individual channels to individual users, i.e. each user is allocated a unique frequency band. These channels are assigned on demand to users who request

© 2005 by Chapman & Hall/CRC

service. For example, in 1983, the first U.S. cellular telephone system — Advanced Mobile Phone System (AMPS), which is based on FDMA — divided 40 MHz of spectrum into 666 duplex channels with a one-way bandwidth of 30 kHz ($666 \times 2 \times 30\,\text{kHz} \approx 40\,\text{MHz}$).

TDMA systems share a single radio-frequency (RF) channel with several users, but divide the radio spectrum into nonoverlapping TS. In each digital TS only one user is allowed either to transmit or receive, and the slots are rotated among the users during a periodic time. Consequently, packet transmission in a TDMA system occurs in a serial fashion, with each user taking turns accessing the channel. For example, at the beginning of 1990s, the first U.S. Digital Cellular (USDC) system made use of TDMA, digital modulation (a type of DQPSK), and speech coding, implementing three times the capacity that of the AMPS. It replaced single-user analog channels with digital channels that can support three users in the same 30 kHz bandwidth.

Besides FDMA and TDMA, a code division multiple access (CDMA) system places all users on the same frequency spectrum at the same time, and use different pseudorandom codes to distinguish between the different users (see also Section 27.3.1 and Figure 27.5). Each code is approximately orthogonal to all other codes. The receiver performs a time correlation to detect only the desired code. The advantage of using CDMA over FDMA and TDMA is that it provides more capacity. There is no absolute limit on the number of users in CDMA, but the channel performance limits the practical capacity of CDMA, which gradually degrades for all users as their number increases. For example, in 1993, a cellular system based on CDMA was developed by Qualcomm, Inc. and standardized by the TIA as an Interim Standard (IS-95). The system supports a variable number of users in 1.25 MHz wide channels. IS-95, discussed below, has been successfully installed in many areas of the world, and it is possible to migrate to third-generation (3G) wireless networks with CDMA2000 to give the higher data rates that are needed for video and data transfer, while retaining compatibility with the existing networks.

Note that CDMA access is a form of spread-spectrum communication, which allows multiple users to share the same frequency band by spreading the information signal for each user over a wide frequency bandwidth (several orders of magnitude greater than the minimum required RF bandwidth). A pseudonoise (PN) or pseudorandom sequence is used as a spreading code to convert a narrowband signal to a wideband noise-like signal before transmission. A PN code is a binary sequence that appears random but can be reproduced in a deterministic manner by the intended receivers. At the receiver, cross-correlation with the correct PN sequence de-spreads the spread-spectrum signal and restores the modulated message in the same narrowband as the original data. By using a group of approximately orthogonal spreading codes, it is possible that the receiver picks out a signal based on its code from many other signals with different spreading codes. That is, up to a certain number of users, interference between spread-spectrum signals using the same frequency is negligible by using different orthogonal codes. The system has been likened to hearing many people in a room speaking different languages. Despite a very high noise level, it is possible to pick out the person speaking your own language, e.g. English. Since many users can share the same spread-spectrum bandwidth without interfering with one another, spread-spectrum systems become bandwidth efficient in a multiple user environment.

There are two main types of spread-spectrum technique: frequency-hopped spread spectrum (FHSS) and direct sequence spread spectrum (DSSS). In an FHSS system, the carrier frequencies of the individual users are varied in a pseudorandom manner within a wideband channel. The digital data are broken into uniform-sized bursts which are transmitted on different carrier frequencies. A DSSS system spreads the baseband data by directly multiplying the baseband data pulses with a PN sequence. CDMA uses DSSS.

One of the advantages in CDMA is that handoff can be made easier and more reliable. Unlike the wireless systems that assign different radio channels during a (hard) handoff, it is possible that spread-spectrum mobiles share the same channel (code) in every cell, since adjacent cells use the same RF band in IS-95. There is a degree of risk with the hard handoff approach because, occasionally, a handoff does not proceed smoothly. The IS-95 system provides soft handoff capacity. Soft handoff is a make-before-break system. When a call is in a soft handoff condition, the two BSs transmit the same

© 2005 by Chapman & Hall/CRC

signal, enabling the mobile to receive the signal via two routes at the same time. The system takes advantage of the moment-by-moment changes in signal strength at each of the two cells to pick out the best signal.

In addition to the soft handoff, IS-95 also supports a hard handoff, which is typically done when (1) moving between BSs served by different switches; (2) encountering BSs of different vendors; (3) encountering BSs that support a different air interface, such as CDMA-to-AMPS.

27.5.5 New-Generation Wireless Networks

First generation (1G) wireless systems use analog signaling (analog FDMA[17]) for the user traffic on the air interface, like AMPS. Second generation (2G) wireless systems (GSM or IS-95) use digital voice coding and digital modulation (such as TDMA, CDMA) for the user traffic. Compared with 1G systems, all 2G systems have improved voice privacy and authentication capability, as well as improved system capacity and signal quality.

Besides voice and short messages, the growing demands for wireless networks providing high-rate data services and better spectrum efficiency are driving the deployment of new wireless technologies.

The aim of 3G wireless systems is to provide a single set of standards that can meet a wide range of wireless applications and provide universal access throughout the world. One standard is called the universal mobile telecommunications system (UMTS) given by the European Telecommunication Standard Institute (ETSI). Three main objectives [3] for UMTS systems are:

1. Full coverage and high data rate.
2. Use of different-sized cells (macrocell: suburban area; microcell: urban area; and picocell: in building) for indoor and outdoor applications, with seamless handover between them.
3. High spectrum efficiency.

Three main components constitute the UMTS system: the core network, the UMTS radio access network, and mobile equipment. The general architecture of 3G wireless networks is illustrated in Figure 27.15.

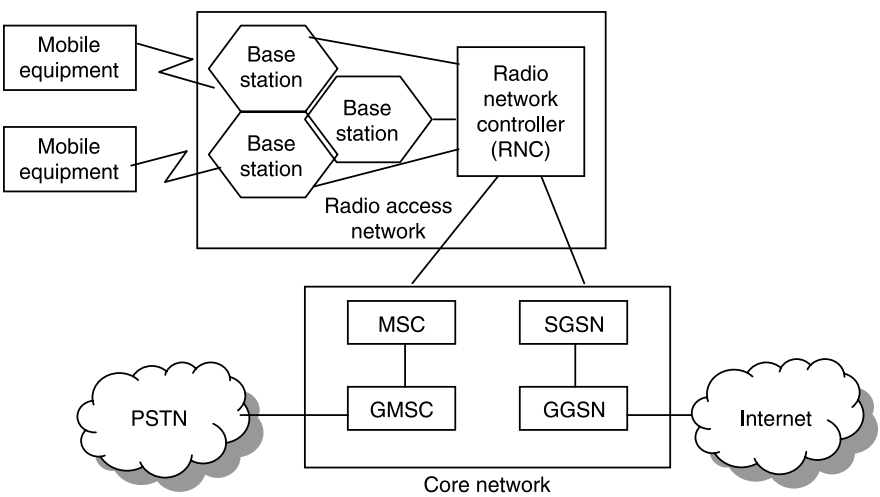


Figure 27.15. The general architecture of 3G wireless networks.

[17]Typically, analog FDMA systems allow a single mobile terminal to occupy a radio channel at a specific time, while the digital FDMA systems allow multiple users to share a single radio channel. The latter is used in 2G wireless systems.

© 2005 by Chapman & Hall/CRC

RNC is the radio network controller for a group of adjacent base stations. It coordinates the overall operation of each BS within its radio subsystem, such as the procedure of handoff decisions. SGSN (the serving general packet radio service node) coordinates the active wireless data devices in its area. GGSN (gateway general packet radio service support node) is a packet switch that routes packets between the UMTS core network and external data network. GMSC (gateway mobile switch center) connects the UMTS network to the public circuit switch networks, such as PSTN. Until 2001, there were three different system specifications for multiple access schemes in 3G wireless systems: wideband CDMA (WCDMA), time division CDMA (TD/CDMA), and CDMA 2000.

27.6 WLANs

Wireless technologies are spanning from wide-area technologies (e.g. satellite-based networks, cellular networks) to local- and personal-area networks. WLANs offer high flexibility and ease of network installation with respect to wired LAN infrastructures. Two main standards for WLANs are IEEE 802.11 and HiperLAN (high performance radio LAN, promoted by ETSI). In this section, we present a brief introduction to IEEE 802.11.

The IEEE 802.11 technology operates in the 2.4 GHz industrial, scientific, and medicine band and provides wireless connectivity for fixed, portable, and mobile stations in a local area. The IEEE 802.11 protocol defines the MAC and physical layer. At the physical layer, two RF transmission methods and one infrared method are defined. The RF transmission standards are FHSS and DSSS. The most popular method is DSSS. Somewhat similar to the 802.3 Ethernet wired line standard, the MAC layer specification for 802.11 uses a protocol scheme known as carrier-sense, multiple access, and collision avoidance (CSMA/CA). 802.11 avoids collisions, instead of the detection of collisions used in 802.3. The MAC layer defines two access methods: the distributed coordination function and the point coordination function. Figure 27.16 illustrates the layer architecture of IEEE 802.11.

The IEEE 802.11 standard defines the protocol for two types of network: infrastructure-based WLAN and an *ad hoc* WLAN. In an infrastructure-based network, there is a centralized controller for each cell,

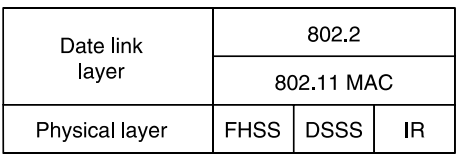


Figure 27.16. IEEE 802.11 layer architecture.

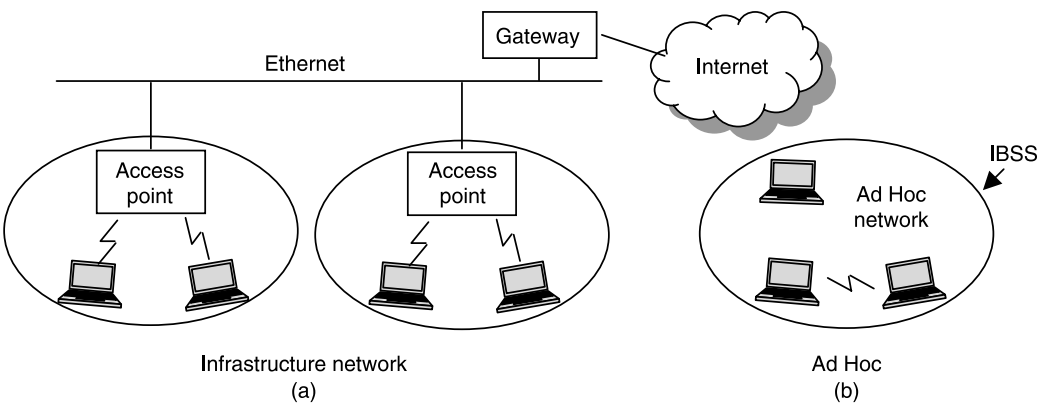


Figure 27.17. Infrastructure and *ad hoc* networks.

© 2005 by Chapman & Hall/CRC

referred to as an access point. Mobile nodes communicate with fixed network access points which are usually connected to the wired network to provide Internet access for mobile devices. In an *ad hoc* network, every 802.11 device in the same cell, or independent basic service set (IBSS) can directly communicate with every other 802.11 device within the cell, without the use of an access point or server. There is no structure to the network; there are no fixed points (see Figure 27.17). In addition, the Bluetooth technology can also be used for building a mobile *ad hoc* network

Currently, a huge amount of new industry standards, new techniques, and new devices are in development. In the near future, more applications and services in wireless networks will have an affect on various aspects of our life.

Acknowledgments

The author acknowledges the support from NSF (grant CCR 0073429) and help from Y. Zhao, during the preparation of this chapter. Examples 27.2 and 27.4, and Figure 27.17 are, respectively taken from bibliographical references [1], [2], and [3].

Bibliography

[1] Walrand, J., *Communication Networks: A First Course*, McGraw Hill, Boston, 1998.
[2] Halshall, F., *Data Communications, Computer Networks and Open Systems*, Addison Wesley, Reading, MA, 1996.
[3] Stojmennovic, I., *Handbook of Wireless Networks and Mobile Computing*, John Wiley, New York, 2002.
[4] Feit, S., *TCP/IP*, 2nd ed., McGraw Hill, New York, 1997.
[5] Halabi, B., *Internet Routing Architectures*, New Riders Publishing, Indianapolis, IN, 1997.
[6] Keshav, S., *An Engineering Approach to Computer Networking: ATM Networks, the Internet, and the Telephone Network*, Addison-Wesley, Reading, MA, 1997.
[7] Thomas, S.A., *IPng and the TCP/IP Protocols*, John Wiley and Sons, New York, 1996.
[8] Stevens, W.R., *TCP/IP Illustrated*, Addison-Wesley, Reading, MA, 1994.
[9] Miller, P., *TCP/IP Explained*, Digital Press, Boston, 1997.
[10] Black, U. and Waters, S., *SONET and T1: Architectures for Digital Transport Networks*, Prentice Hall PTR, New Jersey, 1977.
[11] Seetharam, S.W. *et al.*, A parallel SONET scrambler/descrambler architecture, *1993 IEEE International Symposium in Circuits and Systems*, 2011, 1993.
[12] *Synchronous Optical Network Transport Systems: Common Generic Criterion*, TR-TSY-000253, Bellcore, Morristown, NJ, 1993.
[13] Walrand, J. and Varaiya, P., *High Performance Communication Networks*, Morgan Kaufmann, California, 1996.
[14] Tannenbaum, A.S., *Computer Networks*, Prentice Hall PTR, New Jersey, 2003.
[15] Goralski, W.J., *SONET: A Guide to Synchronous Optical Network*, McGraw Hill, New York, 1997.
[16] Kurose, J.F. and Ross, K.W., *Computer Networking: A Top-Down Approach Featuring the Internet*, Addition Wesley, Boston, 2003.
[17] Rappaport, T.S., *Wireless Communications: Principles and Practice*, Prentice Hall PTR, 1996.
[18] Black, U.D., *Second Generation Mobile and Wireless Networks*, Prentice Hall PTR, 1999.
[19] Harte, L., *3G Wireless Demystified*, McGraw-Hill, 2002.
[20] Garg, V.K. and Wilkes, J.E., *Wireless and Personal Communications Systems*, Prentice Hall PTR, 1996.
[21] Brenner, P., *A technical tutorial on the IEEE 802.11 protocol*, Breeze COM, 1997.
[22] Harte, L., *CDMA IS-95 for Cellular and PCS*, McGraw-Hill, 1999.
[23] Lin, C.-Y., *IS-95 North American standard* — A CDMA based digital cellular system, Columbia University, 1996.

© 2005 by Chapman & Hall/CRC

28

Location-Centric Networking in Distributed Sensor Networks*

Kuang-Ching Wang and Parameswaran Ramanathan

28.1 Introduction

Emerging sensor network technology interconnects spatially distributed sensing devices deployed to monitor and interact with the surrounding physical world [1–3]. Each device (node) in the network senses a limited neighborhood using one or more of its modalities. A single node, however, cannot often sufficiently characterize the physical world due to its limited sensing capability and reliability. Therefore, collaboration among a set of nodes is necessary to produce a complete picture of the area.

Most sensor network applications are *location-centric*, in the sense that they are interested in retrieving information about a certain geographic region as opposed to from a particular set of nodes. This is in contrast to conventional networks, where each node's actions are directed towards a specific set of nodes. Consequently, a fundamentally different approach is needed for efficient information exchange in sensor networks. This approach must address the following challenges.

- *The need for a simple and flexible programming model.* It is a formidable challenge for application designers to develop efficient algorithms if they have to individually manage the addresses, locations, and operational status of a large number of nodes in an area of interest. Moreover, nodes in a sensor network may go inactive to conserve energy, move to a different location, or become nonoperational due to faults. A simple and flexible programming model is needed for application programmers to address nodes in regions of interest easily and to facilitate collaboration among them despite the changes in the network.

*The work reported here is supported in part by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Material Command, USAF, under agreement number F30602-00-2-0555, and by U.S. Army Research grant DAAD19-01-1-0504 under a subrecipient agreement S01-24 from the Pennsylvania State University. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the views of funding agencies.

© 2005 by Chapman & Hall/CRC

- *The need for energy and bandwidth efficiency*. Sensor nodes have limited energy and communication bandwidth. Since wireless devices consume most of their energy in communication, it is a challenge to meet the collaborative needs of an application without significantly draining the energy resources at the nodes.
- *The need for application robustness*. Sensor networks are likely to be deployed in harsh environments. Consequently, a large fraction of nodes may be nonoperational or malfunctioning at any time. Collaborative applications must be robust to these node failures.

At the University of Wisconsin, Madison, we have been developing an approach called *location-centric networking* to address the above challenges. In addition, we have also developed an application programmers interface, called UW-API, and a location-centric *ad hoc* routing protocol called UW-Routing to realize this approach [4–6].

UW-API is motivated by the well-known message-passing interface standard MPI-1.1 [7]. In MPI-1.1, the message passing primitives facilitate information exchange between nodes in a parallel computer system or a network of workstations. In contrast, the message-passing primitives in UW-API are designed for communication between geographic regions. The nodes are not individually addressable. Instead, the addressable entities are geographic regions. Each communication primitive specifies the destination region, and the information exchange takes place between the source and the destination regions. In a dynamic network, the set of nodes participating in a communication primitive adapts to changes in node locations and operational status.

To facilitate region-based communication in UW-API, the underlying routing scheme is location-aware. Several location-aware routing schemes have been studied for *ad hoc* wireless networks [8,9]. In these schemes, a node obtains its geographic location with a global positioning system (GPS) [10]. On-demand location-aided routing (LAR) has been used [6] to utilize the geographic positions of a sender and its receiver to define a constrained zone for route discovery. The flooding route request messages are forwarded and processed only by nodes within this zone for energy and bandwidth conservation. Similarly, the GeoTORA routing protocol was used by Ko and Vaidya [9] to constrain flooding operations in TORA [11] with location information and demonstrates enhanced efficiencies. Nevertheless, they are node-centric and route messages between a pair of nodes. UW-Routing is developed in conjunction with UW-API. UW-Routing is quite different, in that it routes messages between regions rather than nodes. From any node, a message is delivered to most, if not all, nodes in the geographic region addressed. The protocol is an on-demand combining of unicast and constrained broadcast. In general, a message is delivered from a node to a region in two steps. First, the message is unicast to any node in the destination region and then it is flooded to the nodes in the region.

Together, UW-API and UW-Routing provide a location-centric networking approach that addresses the aforementioned challenges. Communication efficiency is addressed in the following two ways: UW-API optimizes the sequence of message exchanges needed to accomplish each of its primitives, and UW-Routing optimizes the number of physical transmissions needed for each message exchange. Application robustness is addressed in the following two ways: first, since every primitive considers a region rather than a particular node, individual node failures are not expected to impact the collaborative computation results significantly; second, time-out constraints are explicitly specified for all primitives so that application performance can be designed to deal with failures in the primitive. To demonstrate the usage of UW-API and UW-Routing, a target tracking application has been developed [5]. The application defines multiple regions and utilizes the collaborative primitives in UW-API to detect, localize, classify, and track ground-vehicle targets along their trajectories. The application is implemented and evaluated on various sensor network testbeds and also software simulators on workstations.

The remainder of this chapter is organized as follows. Section 28.2 describes the location-centric computing model in a distributed sensor network. The network model is defined in Section 28.3. In Section 28.4, we propose the location-centric networking mechanisms UW-API and UW-Routing. The exemplary target tracking application is introduced in Section 28.5. Evaluation studies on sensor network testbeds are summarized in Section 28.6.

© 2005 by Chapman & Hall/CRC

28.2 Location-Centric Computing

The location-centric computing model is based on the premise that sensor network applications are typically interested in acquiring information about a region rather than a particular node. For example, there are queries such as:

- What is the concentration profile of a certain bio-chemical agent in a given area?
- What is the temperature or pressure variation in an area?
- Have there been any unauthorized entries into an area and how many were there?

Oftentimes, such queries cannot be answered by any single node; rather, they require the collaboration among nodes in a certain geographic area to establish a complete picture. This requirement of *location-based* collaboration among nodes does not exist in conventional network applications.

Traditional parallel computing models consider applications designed with a fixed set of nodes. Designers partition the workload among the nodes and define the communication patterns for each node. In such an application, each node serves an indispensable role and the set of nodes is not expected to change throughout the application. The location-centric computing model, however, considers otherwise. The model is not concerned about any particular node; instead, it considers regions to be the entities of interest to most applications. As an application specifies a region of interest, all nodes in the region start to participate in this application. Since the model does not assume the existence of any particular node, it naturally accommodates programming in a dynamic network where nodes may be relocated or become nonoperational. At any given time, a node participates in collaborative computations only if it resides in an active region of an application. Similarly, an application only works with nodes in regions it specifies.

At the University of Wisconsin, Madison, we recently proposed the network application programmers' interface called UW-API [6], which is particularly well suited for location-centric computing. In UW-API, a geographic region plays the role of a node in traditional network interfaces. A region represents a set of nodes residing in a specific geographic area, and it is explicitly *created* and *deleted* with primitives in the API. Currently, we consider a region to occupy a rectangular geographic area, which can be represented with its corner coordinates. A region does not necessarily contains all nodes in the area. Constraints such as sensor types can be used to define regions to be a subset of all nodes in the geographic area. UW-API considers a region to be the only addressable entity in any applications. The API not only supports inter-region communication, it also supports communication within a region. To coordinate communication within a region, it further defines the *manager subregion* of a region. The manager subregion is a subset of nodes in the corresponding region. Nodes in the manager subregion coordinate all collaborative communications in its region.

To understand the location-centric computing model better, consider the following example. Let there be a *controller node* in a distributed sensor network. A temperature monitoring application in this sensor network allows users to retrieve average temperature information from a particular area covered by the network. At the controller node, a user issues the query that specifies an area of interest and expects the answer to be returned. This application can be designed as follows. The controller process (at the controller node) first *creates* a region covering the area of interest. Then, the controller must *send* the query to the created region. As all nodes in the region *receive* the query, a collaborative algorithm is used to compute the average temperature in the region. This is done by all nodes reporting their temperature readings to nodes in the manager subregion, which collect all readings and compute the average. The operation is known as a *reduce* operation in parallel applications. Finally, the computed result is sent back to the controller node.[1] The example demonstrates how a sensor network application can be implemented without the burden of exploiting the network topology or managing individual

[1]Since regions are the only addressable entities in the location-centric computing model, the controller node is not addressable either. We create a *controller region*, to which the controller node belongs. All messages for the controller node are sent toward the controller region.

© 2005 by Chapman & Hall/CRC

nodes. As shown in this example, several communication and collaborative operations are useful in designing sensor network applications. The proposed UW-API perfectly addresses these requirements with its primitives.

28.3 Network Model

We consider a sensor network with a large number of nodes distributed in a large geographic area. Every node is aware of its geographic location and is capable of sensing, information processing, and wireless processing. The radio range of a node is limited, and nodes may communicate in different channels. Thus, it is not necessarily possible for a node to communicate directly with all other nodes in the network. For a node to communicate with nodes other than its immediate neighbors, multihop forwarding is needed. The network is, as a result, a wireless multihop *ad hoc* network.

28.4 Location-Centric Networking

UW-API provides application programmers with a set of primitives to design their collaborative applications in a distributed sensor network. Section 28.4.1 introduces six primitives defined in the current API. These primitives are supported by the underlying location-aware routing scheme called UW-Routing. The routing scheme efficiently delivers a message from a node to a region. Section 28.4.2 presents the UW-Routing protocol in detail.

28.4.1 UW-API

Currently, six primitives are defined in UW-API. For region management, two primitives are defined as *SN_CreateRegion* and *SN_DeleteRegion*. For message passing among regions, *SN_Send* and *SN_Recv* are defined. Two primitives for collaborative computing are defined: *SN_Reduce* allows nodes in a region to aggregate individual information into defined forms such as average, minimum, and maximum values; *SN_Barrier* provides the mechanism to synchronize program execution of all nodes in the same region.

28.4.1.1 SN_CreateRegion

The primitive prototype is as follows:[2]

RegionID = *SN_CreateRegion*(*SourceRegionID*, *Range*, *ManagerRange*, *SensorType*, *Timeout*)

SN_CreateRegion creates a region in *Range* with its manager subregion in *ManagerRange*, where *Range* and *ManagerRange* are corner geographical coordinates of the square areas. A region can be constrained to have only nodes with the specified *SensorType*. The caller of the primitive must specify its own region handle *SourceRegionID*. Once the region is successfully created, an acknowledgement will be sent by the just created manager subregion to the source region. *Timeout* specifies the amount of time the caller is willing to wait for the acknowledgment before the call returns as failed. Once created, a region is assigned an integer handle *RegionID*.

28.4.1.2 SN_DeleteRegion

Status = *SN_DeleteRegion*(*SourceRegionID*, *RegionID*, *Timeout*)

SN_DeleteRegion deletes the region corresponding to handle *RegionID*. The caller of the primitive also specifies its own region handle *SourceRegionID*. Once the region is successfully deleted, the nodes in the manager subregion of the deleted region send an acknowledgement to the source region and the call

[2]Types of variables are omitted in the following presentation. Interested readers may refer to Ramanathan *et al.* [6].

© 2005 by Chapman & Hall/CRC

returns. *Timeout* specifies the maximum amount of time the call waits for the acknowledgement before it returns with *Status* as failure.

28.4.1.3 SN_Send

Status = *SN_Send*(*SourceRegionID*, *DestinationRegionID*, *Data*, *DataType*, *Size*, *Tag*)

SN_Send delivers a message named *Data* from the caller node to a specified region. The regions are refered to by their handles *SourceRegionID* and *DestinationRegionID*. *DataType* and *Size* specify the type and size of the message. The *Tag* provides auxiliary information, such as data encoding options, and application types. *SN_Send* returns with *Status*, successful or not, once the message is handed to the routing agent. *SN_Send* does not guarantee delivery to the destination region.

28.4.1.4 SN_Recv

Status = *SN_Recv*(*SourceRegionID*, *Data*, *DataType*, *Size*, *Tag*, *Timeout*)

SN_Recv is the matching primitive that receives a message sent with *SN_Send*. It receives a message named *Data* with *DataType* and *Size* from a specified region with its handle *SourceRegionID*. If *Tag* is specified, it further constrains the scope of messages to be received. Both *SourceRegionID* and *Tag* can be set to accept any incoming messages. *Timeout* specifies the amount of time the caller is willing to wait for a matching message to be returned. If no matching message is received before it times out, then the call returns null and the *Status* indicates failure.

28.4.1.5 SN_Reduce

The primitive prototype is as follows:

Status = *SN_Reduce*(*RegionID*, *Data*, *DataType*, *Count*, *Operation*, *NumNodes*, *Timeout*, *Tag*)

In parallel computing terminology, *reduce* is a collective communication primitive that collects data from all nodes in a specified group and performs an operation such as sum, min, or max on all collected data. The same collective communication is supported in UW-API with *SN_Reduce*. When an application needs to perform a reduce operation, it programs all nodes in the specified region to call this primitive. Each participating node delivers its *Data*, which is an array of *DataType* with *Count* elements, to nodes in the manager subregion and the specified *operation* is performed on all collected data. All the nodes in the manager subregion perform this data collection and computation. Fault-tolerant information fusion techniques [12] can be used to ensure that all the nodes in the manager subregion arrive at the same result even in the presence of faulty nodes. Results may differ among these nodes and some may be faulty in a harsh environment. To enhance robustness and fault tolernace of the computation, there exist sophisticated algorithm. The call termination depends on *NumNodes* and *Timeout*. Nodes in the manager subregion are blocked until at least *NumNodes* reports are collected or a *Timeout* expires. All other nodes return after sending their data. Again, *Tag* provides auxiliary information for the operation if necessary.

28.4.1.6 SN_Barrier

Status = *SN_Barrier*(*RegionID*, *NumNodes*, *Timeout*, *Tag*)

In parallel computing terminologies, *barrier* is a collective communication primitive that synchronizes a group of processes at a certain point in an application. When it calls the primitive, a process blocks until all processes in the group have called the primitive, i.e. arrived at the pre-specified points in the

© 2005 by Chapman & Hall/CRC

application. For processes to know how many peer processes have arrived, each process broadcasts a message in the group once it calls the primitive. In conventional parallel applications, the number of processes N is statically known. Thus, all processes complete the call at hearing the Nth broadcast.

In UW-API, *SN_Barrier* synchronizes all nodes in a specified region. Similarly, each node gets blocked and broadcasts a message as they call *SN_Barrier*. They also count the number of broadcast messages to determine the call termination. However, in a distributed sensor network, the number of nodes in a region can not be known *a priori*. Furthermore, the number may change as nodes relocate, enter a sleep mode, or expire. Similar to *SN_Reduce*, *NumNodes* and *Timeout* are speficied for call termination. Unlike *SN_Reduce*, *SN_Barrier* blocks all nodes until at least *NumNodes* broadcasts are heard or a *Timeout* period has elapsed. *Tag* provides auxiliary information for the operation if necessary.

28.4.2 UW-Routing

The UW-Routing protocol provides region-based communication that facilitates UW-API. As an on-demand protocol, UW-Routing delivers a message to a specific region in two steps. First, it *unicasts* the message to an arbitrary node in the specified region. Once the message arrives at the region, the second step is to broadcast the message to all nodes in the region.

UW-Routing uses message flooding in various places. For energy and bandwidth efficiency, all flooding operations are geographically constrained. Similar to Gropp *et al.* [7], each flooded message has a square *forwarding zone* specified in geographic coordinates. The forwarding zone must cover the source and destination regions. Figure 28.1 shows our strategy in deciding a messages forwarding zone. The controllable margin accomodates situations in which feasible routes do not exist in a minimum forwarding zone. A node receiving a flooded message continues the forwarding if and only if it resides within the message's forwarding zone. The following describes the message formats, route discovery for a remote region, and routing within a region.

28.4.2.1 Message Formats and Address Resolution

Figure 28.2 summarizes the message formats. Each message serviced by UW-Routing carries a common header containing its message identification number, message type, network address of previous forwarder, its forwarding zone, and a time stamp. For each message type, a specific extended header follows. Four message types are defined: route request (RREQ), route reply (RREP), manager route construction (MRC), and payload.

RREQ and RREP messages are used in route discovery. They have the same extended header containing the geographic coordinates, as well as their region handles for the source and destination regions. Geographic region coordinates are explicitly specified only in RREQ, RREP, and the payload

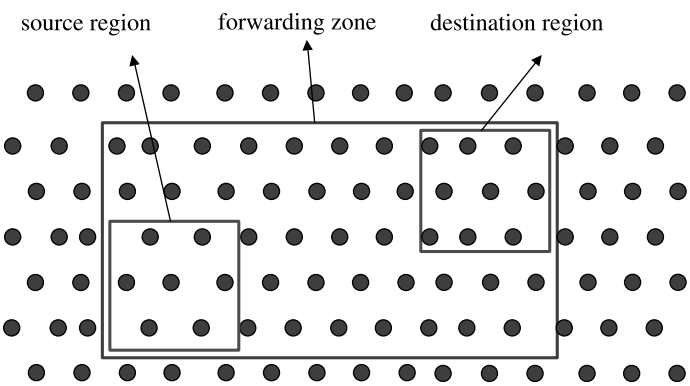


Figure 28.1. A message-forwarding zone covering source and destination regions.

© 2005 by Chapman & Hall/CRC

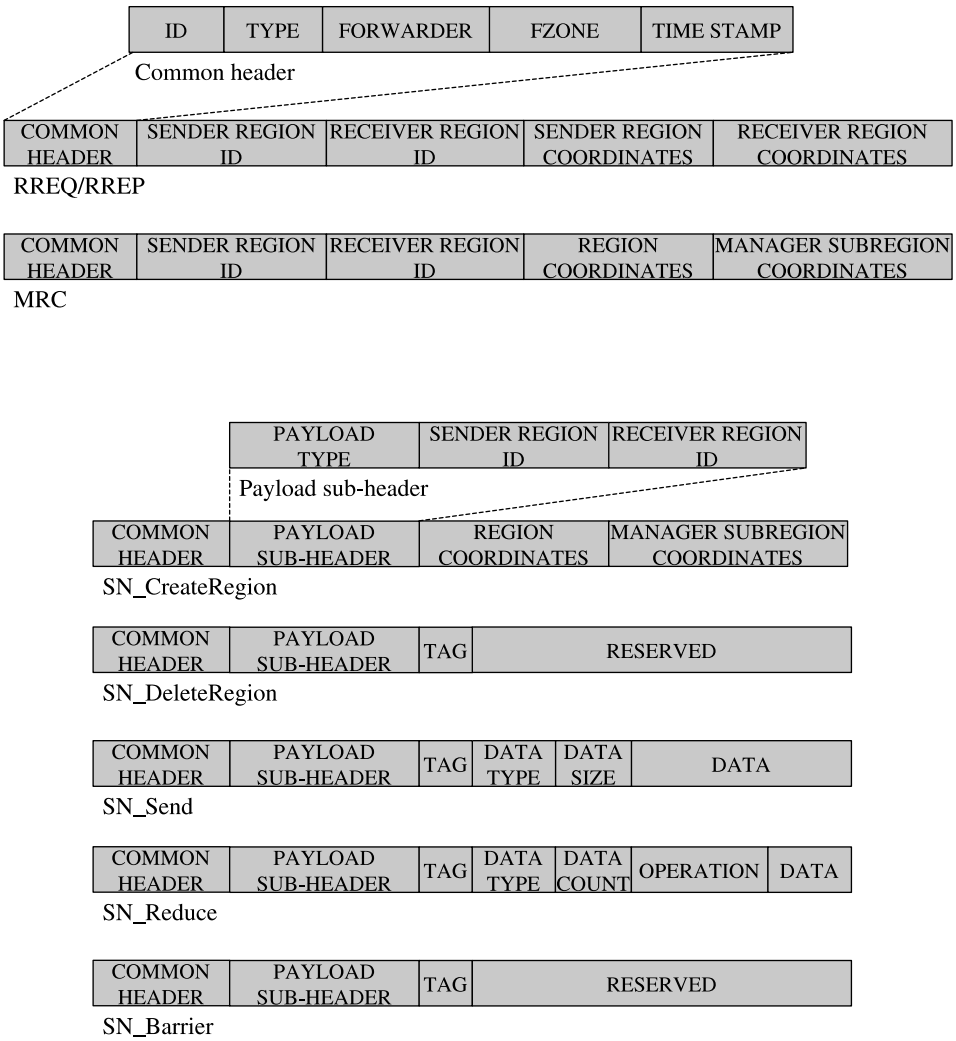


Figure 28.2. Message formats for UW-Routing.

for *SN_CreateRegion*. Once the route discovery and creation completes for a region, further routing for the region is based on the region handle instead of geographic coordinates. An MRC message is used in constructing routes in a region to facilitate communication between manager and nonmanager nodes. Its extended header carries the handle of the respective region. Payload messages associated with respective UW-API primitives are also shown in Figure 28.2 and are self-explanatory according to their functionalities.

Network addresses must be associated with corresponding *link layer addresses* before messages can be physically transmitted through a node's wireless interface. The address resolution is done in conjunction with route discovery with no additional overhead. Note that each message carries the forwarding node's network address. It is assumed that the link layer header carries link layer addresses of both sending and receiving nodes. During route discovery, RREQ and RREP are flooded. By associating the network and link layer addresses of the sender of each message, address resolution for all neighboring nodes is done after one flooding operation. As a node potentially has more than one wireless interface, an interface is associated with each neighbor as well. Table 28.1 shows one example of an address resolution table.

© 2005 by Chapman & Hall/CRC

Table 28.1. An address resolution table for UW-Routing

Entry number	Node network address	Link layer address	Interface
1	1	10.0.1	A
2	7	10.0.23	B
3	5	10.0.29	B
4	12	10.0.42	A
5	15	10.0.37	B
...	...	...	...

Table 28.2. A routing table for UW-Routing

Entry number	Region ID	Coordinates	Next hop node	TTL
1	1	[0, 0, 10]	7	70
2	1379	[10, 20, 70]	7	25
3	6517	[25, 75, 75]	5	82
4	7482	[250, 10, 350, 110]	12	13
5	1010	[100, 150, 200, 250]	15	2
...	...	...	...	...

28.4.2.2 On-Demand Route Discovery

Each router maintains a routing table, as illustrated in Table 28.2.

For each region with a known route, the table has an entry recording the network address of the next-hop node of the route. The *Timeout* for each entry indicates the amount of time it remains valid and an expired route is removed from the table. When the router receives a message whose destination region does not exist in the table, the message is withheld in a queue and the route discovery procedure is initiated. The route discovery is based on RREQ and RREP messages.

The process starts with the router creating and flooding an RREQ message in the network. As mentioned earlier, the flooding is constrained within its forwarding zone. If a node receives an RREQ and it currently has a valid route for the requested region, then the node generates an RREP and floods it back to the RREQ source region. If a node receives an RREQ but does not already know of the region, then it simply continues the flooding. If a node that belongs to the requested region receives the RREQ, then it also generates an RREP and floods it back to the RREQ source region.

The routing table is updated at each node by monitoring RREQ and RREP messages. When an RREQ is received and the source region is not known, the source region is added to the table with the RREQ forwarding node as its next hop. When an RREP is received and its source region (that sends the RREP) is not known, the source region is also added to the table with the RREP forwarding node as its next hop. Whenever a new entry is added, the router inspects all messages in the queue and services those messages whose destination region is now known to the node.

28.4.2.3 Routing within a Region

Within a region, two communication patterns frequently arise. First, it is necessary to disseminate messages from any node to all other nodes in the region. This occurs when a message arrives at a node in the destination region and the node must disseminate the message to all other nodes in the region. This essentially requires flooding a message to all nodes in a region. UW-Routing floods a message within a region if it is destined for all nodes in the region.

The second communication pattern is to collect messages from each node to nodes in the manager subregion. This occurs as *SN_Reduce* is called and all nodes report their data messages to the manager nodes. In this case, flooding certainly is not the most efficeint solution. Instead, UW-Routing constructs

© 2005 by Chapman & Hall/CRC

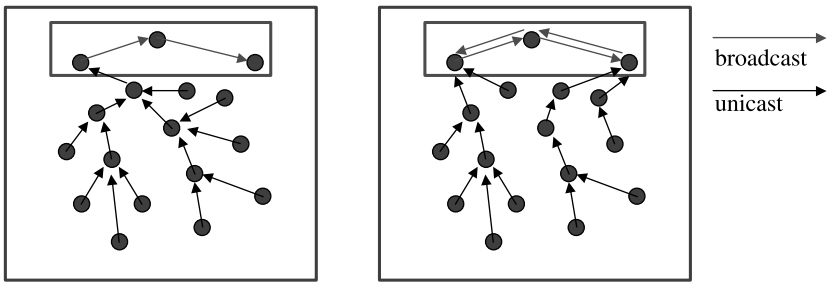


Figure 28.3. Message delivery trees rooted at nodes in a manager subregion.

a tree that roots at one of the manager nodes and spans all other nodes in the region. Thus, all *SN_Reduce* messages are delivered to the manager node following the tree structure. The manager node then floods all messages it receives within the manager subregion such that all magaer nodes receive the same set of messages.

To construct a tree, a manager node floods an MRC message in the region after the region is created and a random time period has elapsed. The random wait is to reduce the probability of more than one node initiating the tree construction. Thus, a manager node receiving an MRC message will no longer initiate another tree construction itself. A node receiving an MRC message records the forwarding node as its parent in the tree and continues flooding the message. A tree is constructed once all nodes determine their parents. Still, there can potentially be more than one manager node constructing multiple trees in the region. This is equally acceptable as long as each node belongs to one tree. Therefore, each node accepts the first MRC message it receives and joins the respective tree. Figure 28.3 illustrates two such tree structures.

28.5 Target Tracking Application

To demonstrate the location-centric computing approach for collaborative application designs in a distributed sensor network, a target tracking application has been developed at the University of Wisconsin with UW-API and UW-Routing. Figure 28.4 shows a typical sensor network where we detect and track potential target activities. In this application, the sensor network is tasked to detect the presence of a certain type of ground vehicle and to track its movement through the sensor field. With UW-API, regions are first created around potential target entry areas. Only devices in these initial regions are active before any taget enters the area. All other devices wait passively to be activated by these initial regions when needed. In an active region, the following operations are performed:

- *Per-node detection*: each device constantly attempts to detect the presence of a target with its own sensing modalities. Detection decisions of each node are periodically reported to its manager subregion.
- *Region detection*: each device in the manager subregion collects detection decisions from all nodes in the region. Individual decisions are combined with robust information fusion algorithms to arrive at a consensus decision for the region as a whole.
- *Per-node classification*: when a device decides that a target is present, it further attempts to determine the target vehicle type with its classifier algorithm. Classification results of each node are periodically reported to its manager subregion.
- *Region classification*: each device in the manager subregion collects classification results from all nodes in the region. Individual results are combined to arrive at a consensus classification result for the region and whether the vehicle is of the desired type.
- *Target localization*: once region detection and region classification confirm the presence of a desired type of target, the target must be localized. This is done collaboratively with all nodes in the region using an energy-based localization algorithm [13]. In this algorithm, all nodes report

© 2005 by Chapman & Hall/CRC

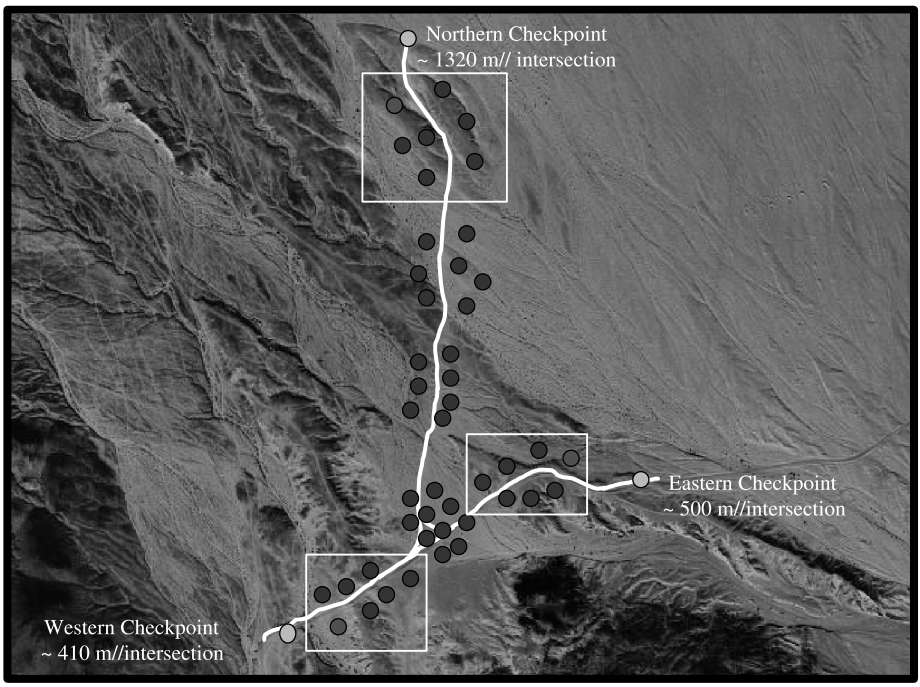


Figure 28.4. A typical sensor network for the target tracking application.

to nodes in the manager subregion with their energy measurements, based on which the target location is estimated.

- *Target tracking*: with a series of estimated target locations at consecutive time instances, nodes in the manager subregion are able to estimate the target speed and direction. Based on this information, near-term target locations can be estimated. If the predicted target location lies beyond the current region, then additional regions are to be created along potential future target trajectories and the regions are to be tasked with this same tracking application.

Figure 28.5 shows the software architecture. The shaded portions are location-centric components that interact with wireless communication interfaces and data peripherals on each device through

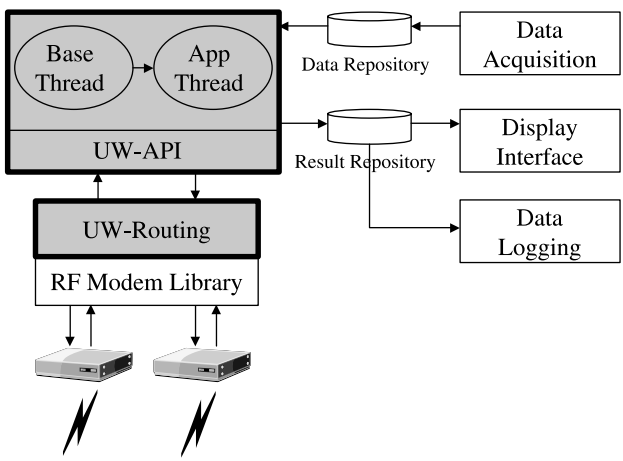


Figure 28.5. The UW software architecture for location-centric computing.

© 2005 by Chapman & Hall/CRC

UW-API and UW-Routing. Applications are managed with threads. On each device, the base thread is responsible for application creation, termination, and accepting incoming messages from the router. Each application is designed as a thread. When a device receives a message commanding it to initiate an application, the base thread spawns the corresponding thread. Through UW-API, an application may design its collaboration within each region, as well as among multiple regions. In our tracking application, various UW-API primitives are used to facilitate both intra-region and inter-region collaborations.

Figure 28.6 shows the pseudocode of the tracking application, consisting of six software modules corresponding to the aforementioned operations.

- NodeDet: each device runs a constant false alarm rate (CFAR) energy detector based on its own sensor measurements in NodeDet. Periodically, all nodes call *SN_Reduce* with its summing operation on their binary detector decisions. The sum of all decisions is thus computed at nodes in the manager subregion for later use in the DetFus module. At the same time, each node uses *SN_Send* to report the average energy detected during the period to nodes in the manager subregion for later use in the TarLoc module.

```
Nodes in manager sub-region:

For every interval of T,
Manager_track(vehicle_type){
    //DetFus
        SN_Reduce(my_region, detection_result);
    //ClassFus
        SN_Reduce(my_region, classification_result);
    //Collect energy reports from nodes
        report_deadline=set_report_deadline();
        while current_time<=report_deadline,
            SN_Recv(node_energy);
            energy_reports=energy_reports ∪node_energy;
    //TarLoc and TarTrak if desired type of target exists
    if detection_result==True and classification_result==vehicle_type,
        Location_estimate=TarLoc(energy_reports);
        Predicted_location=TarTrak(Location_estimate);
        if Predicted_location is beyond this region,
            New_region=Predict_new_region(Predicted_location);
            SN_CreateRegion(New_region);
}

    Nodes not in manager sub-region:

    For every interval of T,
    Nonmanager_track(){
        //NodeDet
            detection_result=NodeDet(sensor_reading);
        //NodeClass
            classification_result=NodeClass(sensor_reading);
        //DetFus
            SN_Reduce(my_region, detection_result);
        //ClassFus
            SN_Reduce(my_region, classification_result);
        //Send energy report to manager sub-region
            SN_Send(my_manager_region, node_energy);
    }
```

Figure 28.6. Pseudocode of the target tracking application.

© 2005 by Chapman & Hall/CRC

- DetFus: at nodes in the manager subregion, the reduced sums of detector decisions are used in a robust fusion algorithm to arrive at consensus region detection decisions [12].
- NodeClass: if NodeDet decides a target is present, then NodeClass is invoked to classify the type of target with its detector energy series. Each node uses *SN_Send* to periodically report the classifier result to nodes in the manager subregion.
- ClassFus: periodic reports of classifier results are collected with *SN_Recv* at nodes in the manager subregion. A robust fusion algorithtm [14] is used to aggregate these reports into a consensus whether the target is of the desired type.
- TarLoc: periodic energy reports are collected with *SN_Recv* at nodes in the manager subregion. If DetFus decides there is a target and ClassFus confirms the desired type, then the energy reports are used to estimate the target location [13].
- TarTrak: this tracks a confirmed target by recording its consecutive location estimates and predicting its near-term future positions. This is done at nodes in the manager subregion. If the predicted location lies beyond the current region, then *SN_CreateRegion* is used to create a new region around the predicted location. Finally, a command message is sent with *SN_Send* to the new region to initiate the same application in the new region.

28.6 Testbed Evaluation

UW-API, UW-Routing, and the target tracking application have been implemented on dedicated sensor devices developed for SensIT. These devices are equipped with acoustic, seismic, and infrared sensors. They are also equipped with GPS and wireless communication interfaces. To evaluate the location-centric approach, we deployed these devices in two testbeds and measured the network performances. The two testbeds are: a testbed of 20 nodes deployed in Waltham, Massachusetts; and a testbed of 70 nodes deployed in 29 Palms, California. The testbed topologies are shown in Figure 28.7. In these testbeds, the tracking application is launched in three scenarios and the network performance is evaluated. The three scenarios are:

- Cross-run. In this scenario, the network tracks two vehicles Armored Attack Vehicle (AAV) and Dragon Wagon (DW) at the same time as they cross each other in the 20-node testbed.

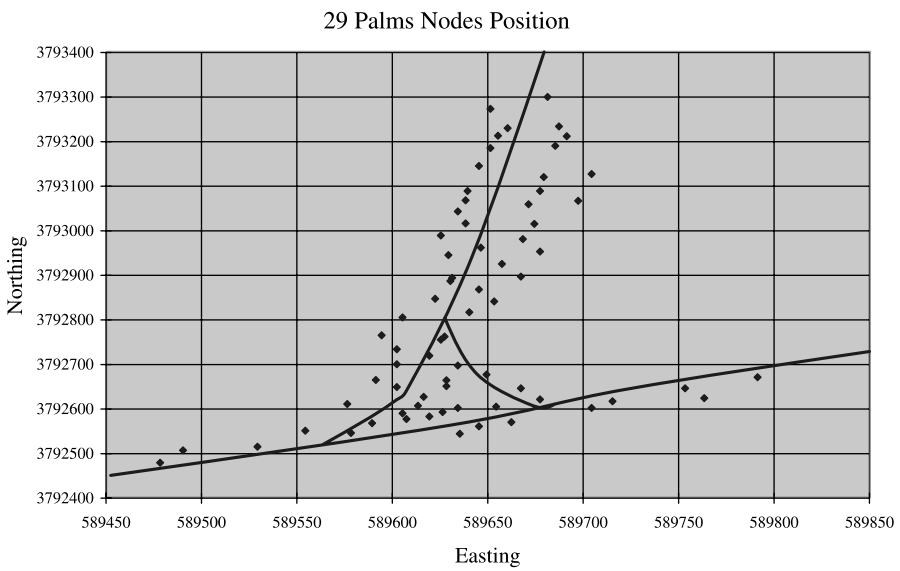


Figure 28.7. Sensor network testbed topologies.

© 2005 by Chapman & Hall/CRC

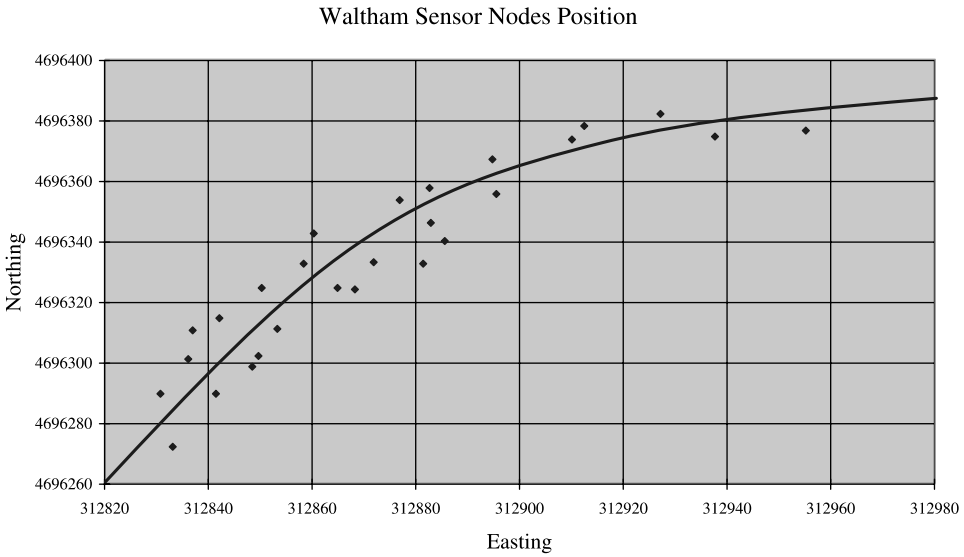


Figure 28.7. Continued.

- Turn-back. In this scenario, the network tracks two vehicles (AAV and DW) at the same time as they meet and turn back in the 20-node testbed.
- Single-AAV. In this scenario, a single AAV runs from east to west in the 70-node testbed.

The network performace is evaluated in the following aspects:

- control messages versus payload messages forwarded
- in-region messages versus out-of-region messages forwarded
- overall per-node bandwidth consumption

28.6.1 Control Messages versus Payload Messages

Location-centric applications require control message exchanges only during region creation and initial setup stages. Figure 28.8 summarizes the number of control and payload messages ever transmitted by each node in all scenarios. All messages have the same size of 88 bytes. The number of control messages is far smaller than that of payload messages, indicating the relatively low control overhead required in a location-centric application.

28.6.2 In-Region Messages versus Out-of-Region Messages

Location-centric applications are expected to have primarily localized computation and communication. We analyze for each node the number of transmitted messages destined for its own region (in-region) versus those destined for a foreign region (out-of-region). Figure 28.9 summarizes the comparison results for three scenarios. As expected, the fraction of out-of-region messages serviced by each node is low.

28.6.3 Overall Per-Node Bandwidth Consumption

Figure 28.10 shows the typical bandwidth consumption of a manager node in each scenario. The bandwidth profiles are actually similar among the scenarios. All of them start with a short surge. After the surge, they stay low for a certain amount of time before the final phase begins, where bandwidth consumption rises and falls in a periodic pattern. The profile closely reflects the tracking application's

© 2005 by Chapman & Hall/CRC

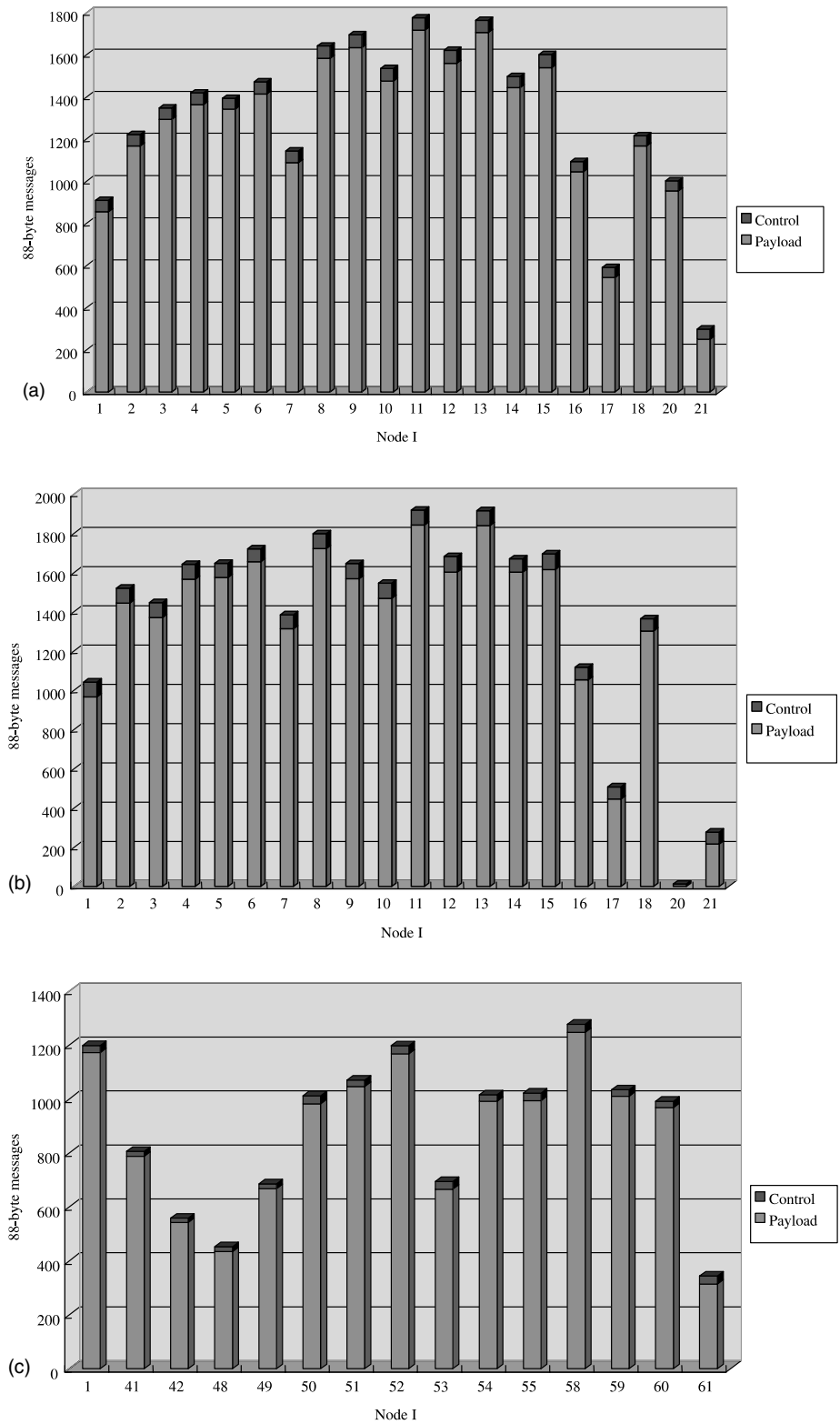


Figure 28.8. The number of control and payload messages transmitted by each node in the three scenarios: (a) cross-run; (b) turn-back; (c) single-AAV.

© 2005 by Chapman & Hall/CRC

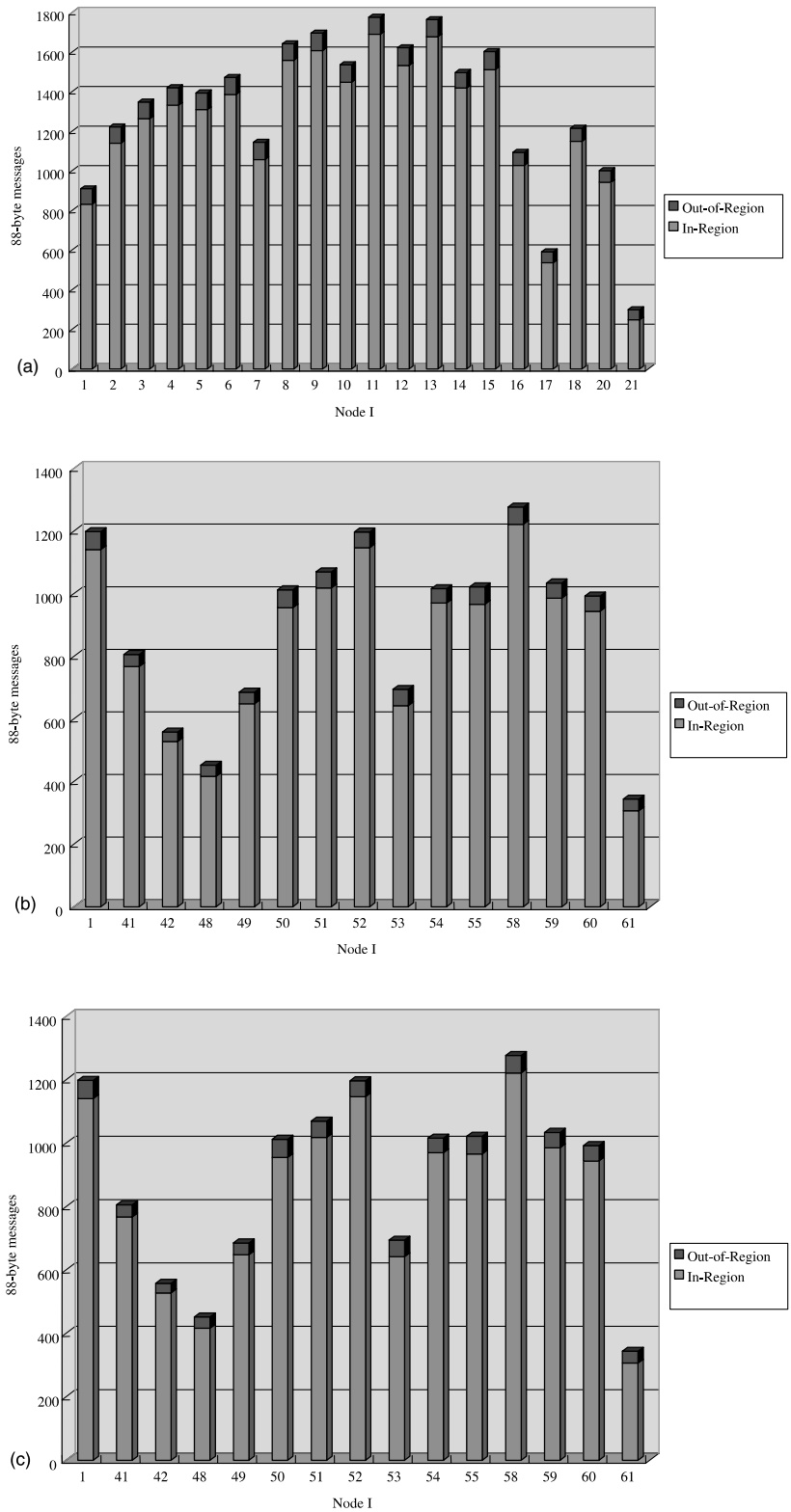


Figure 28.9. The number of in-region and out-of-region messages transmitted by each node in the three scenarios: (a) cross-run; (b) turn-back; (c) single-AAV.

© 2005 by Chapman & Hall/CRC

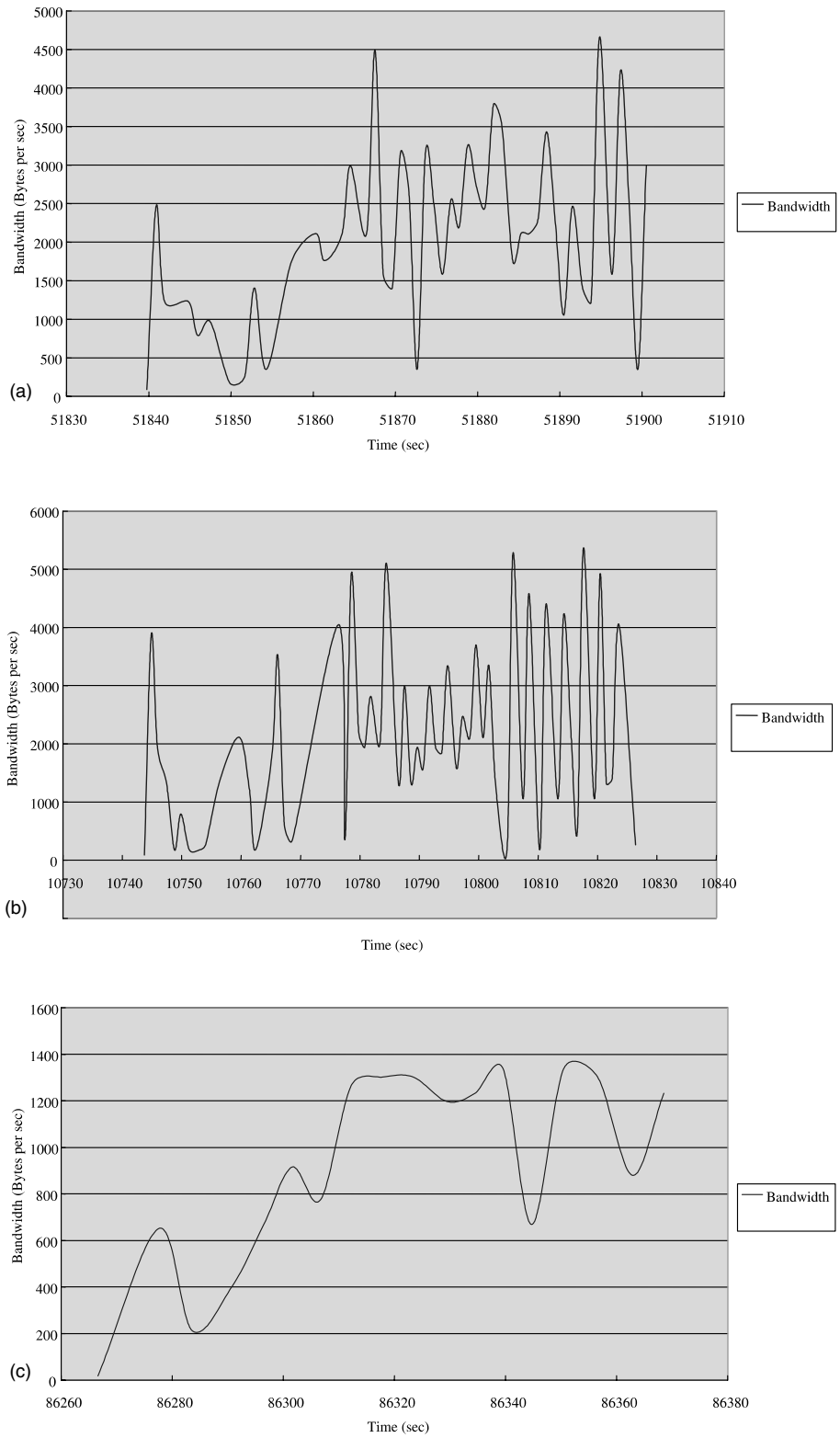


Figure 28.10. Bandwidth consumption of a node in the manager subregion in the three scenarios: (a) cross-run; (b) turn-back; (c) single-AAV.

© 2005 by Chapman & Hall/CRC

communication pattern. The initial surge indicates control message exchanges involved in region creation. Once a region is created, nodes remain inactive if no tasks are issued. As soon as a region is tasked with the tracking application, the collaborative algorithm requires nodes in a region periodically collect detection and classification decisions in the manager subregion. The periodic bandwidth profile is thus explained.

Reference

[1] Agre, J. and Clare, L., An integrated architecture for cooperative sensing networks. *Computer*, 33, 106, 2000.

[2] Estrin, D. *et al.*, Instrumenting the world with wireless sensor network. In *Proceedings of ICASSP 2001*, 2001, 2675.

[3] Kumar, S. *et al.* (eds), Special issue on collaborative signal and information processing in microsensor networks. *IEEE Signal Processing Magazine*, March, 19(2), 13–14, 2002.

[4] Brooks, P. *et al.*, Distributed target classification and tracking in sensor networks. In *Proceedings of IEEE*, 2003.

[5] Ramanathan, P., Location-centric approach for collaborative target detection, classification, and tracking. In *Proceedings of IEEE CAS Workshop on Wireless Communication and Networking*, September 2002.

[6] Ramanathan, P. *et al.*, UW-API: a network routing application programmer's interface. Technical Documentation for DARPA SensIT Program, http://www.ece.wisc.edu/~sensit, October 2001.

[7] M. Snir and Gropp, W. *et al.*, *MPI — The Complete Reference*, Vol. 2, MIT Press, 1998.

[8] Ko, Y.-B. and Vaidya, N.H., Location-aided routing (LAR) in mobile *ad hoc* networks. In *Proceedings of ACM MOBICOM 1998*, October 1998, 66.

[9] Ko, Y.-B. and Vaidya, N.H., GeoTORA: a protocol for geocasting in mobile *ad hoc* networks. In *Proceedings of ICNP*, 2000, 240.

[10] Parkinson, N. and Gilbert, S., NAVSTAR: global positioning system — ten years later, In *Proceedings of IEEE*, 1983, 1177.

[11] Park, V.D. and Corson, M.S., A highly adaptive distributed routing algorithm for mobile wireless networks. In *Proceedings of INFOCOM*, April 1997, 1405.

[12] Clouqueur, T. *et al.*, Value-fusion versus decision-fusion for fault-tolerance in collaborative target detection in sensor networks. In *Proceedings of International Conference on Information Fusion*, 2001.

[13] Li, D. *et al.*, Detection, classification, tracking of targets in micro-sensor networks. *IEEE Signal Processing Magazine*, March, 17–19, 2002.

[14] D'Costa, A. and Sayeed, A., Collaborative signal processing for distributed classification in sensor networks. In *Proceedings of International Workshop on Information Processing in Sensor Networks*, 2003.

© 2005 by Chapman & Hall/CRC

29

Directed Diffusion*

Fabio Silva, John Heidemann, Ramesh Govindan, and Deborah Estrin

29.1 Introduction

Traditional sensing models assume one or a few powerful sensors and centralized computation. Today, technological trends enable the creation of inexpensive, small, intelligent devices for sensing and actuation. If many small sensors can work together as a *sensor network*, then they provide several advantages over traditional centralized sensing. By placing the sensor close to the object being sensed, signal processing and target discrimination problems in sensing can be greatly simplified. By communicating over several short hops rather than one long hop, the energy consumed in communication can be reduced [1]. Moreover, by processing data *in* the network, often the amount of data transfered can be reduced, saving further energy [2].

Motivated by robustness, scaling, and energy efficiency requirements, this chapter examines a new data dissemination paradigm for such sensor networks. This paradigm, which we call *directed diffusion*,[1] is data centric. Data generated by sensor nodes is named by attribute–value pairs. A node requests data by sending *interests* for named data. Data matching the interest is then ''drawn'' down towards that node. Intermediate nodes can cache, or transform, data and may direct interests based on previously cached data (Section 29.3).

Directed diffusion is significantly different from IP-style communication, where nodes are identified by their end-points, and inter-node communication is layered on an end-to-end delivery service provided within the network. In directed diffusion, nodes in the network are application-aware as we allow application-specific code to run in the network and assist diffusion in processing messages. This allows directed diffusion to cache and process data in the network (aggregation), decreasing the amount of end-to-end traffic, and resulting in higher energy savings. We show that by using directed diffusion one can realize robust multi-path delivery, empirically adapt to a small subset of network paths, and achieve significant energy savings when intermediate nodes aggregate responses to queries (Section 29.5).

*This work was supported by DARPA under grant DABT63-99-1-0011 as part of the SCAADS project.

[1]Van Jacobson suggested the concept of ''diffusing'' attribute named data for this class of applications that later led to the design of directed diffusion.

© 2005 by Chapman & Hall/CRC

This chapter describes diffusion, starting from the point of view of an application (Section 29.2) and naming (Section 29.2.2). We realize these abstractions with lower-level primitives and several different data dissemination algorithms described in Section 29.3, and show how applications can influence routing (Section 29.4). We summarize simulation and experimentation results in Section 29.5.

29.2 Programming a Sensor Network

The innovations of diffusion are approaches to allow applications to process data as it moves through the network, and dissemination algorithms that select efficient paths through the network. Although these topics are important and we explore them later in the chapter, applications require abstractions over these details. This section presents an *application-level* view of diffusion, looking at our publish/subscribe-based API and how applications name data in the network.

29.2.1 The Publish/Subscribe API

We have adopted a publish/subscribe-based API for diffusion, shown in Figure 29.1.[2] To receive data, users or programs *subscribe* to a particular set of attributes, becoming data *sinks*. A callback function is then invoked whenever relevant data arrives at the node. Sensors *publish* data that they have, becoming data *sources*. In both cases, what data are provided or received are described by an attribute-based naming scheme described in Section 29.2.2. It is the job of the diffusion dissemination algorithms (Section 29.3) to ensure that data are communicated efficiently from sources to sinks across a multi-hop network. In general, publishing and subscribing sends messages across the network. The exact cost of these operations depends on which diffusion algorithm is used.

To allow applications to influence data as it moves through the network, users can create *filters* at each sensor node with the filter APIs at the bottom of Figure 29.1. Filters indicate what messages they are interested in by attributes; each time a matching message arrives at that node the filter is allowed to inspect and alter its progress in any way. Filters can suppress messages, change where they are sent next, or even send other messages in response to one (perhaps triggering further sensors to satisfy a query).

A more complete reference to directed diffusion APIs and example code is available in the diffusion manual [4].

Publish/Subscribe APIs:

```
handle NR::subscribe(NRAttrVec *subscribe_attrs, const NR::Callback * cb);
int NR::unsubscribe(handle subscription_handle);
handle NR::publish(NRAttrVec *publish_attrs);
int NR::unpublish(handle publication_handle);
int NR::send(handle publication_handle,
             NRAttrVec *send_attrs);
```

Filter-specific APIs:

```
handle NR::addFilter(NRAttrVec *filter_attrs, u_int16_t priority, FilterCallback *cb);
int NR::removeFilter(handle filter_handle);
void NR::sendMessage(Message *msg, handle h, u int16 t priority = 0);
```

Figure 29.1. Basic diffusion APIs for sending and receiving data, and for adding filters.

[2]This API was originally designed in collaboration with Dan Coffin and Dan van Hook [3]; we have since extended it to support filters.

© 2005 by Chapman & Hall/CRC

29.2.2 Naming Concepts

Diffusion uses an attribute-based naming scheme to associate sources and sinks and to trigger filters. This flexible approach to naming is important in several ways. First, attribute-based naming is consistent with the publish/subscribe application-level interface (Section 29.2) and many-to-many communication. Diffusion's naming scheme is *data-centric*, allowing applications to focus on what data is desired rather than on individual sensor nodes. The approach also supports multiple sources and sinks, rather than simple point-to-point communication. Thus applications may subscribe to "seismic sensors in the southeast region" rather seismic sensors #15 and #35, or hosts 10.1.2.40 and 10.2.1.88.

Second, diffusion attributes provide some structure to a message. By identifying separate fields, data dissemination algorithms can use application data to influence routing. For example, application-specific geographic information can limit where diffusion must look for sensors. In addition, treating messages as sets of attributes simplifies application and protocol extensions (a need also suggested for future Internet-based protocols [5]).

Finally, attributes serve to associate messages with sources, sinks, and filters via *matching*. If the attributes in a sink's subscription match those of a source's publication, then diffusion must send any published data to the sink.

29.2.3 Matching in Naming

Each set of attributes in diffusion is a set of (key, type, operator, value) tuples. The most important parts of an attribute are the *key* and *value*, which together specify the meaning of the data (longitude, temperature, detection confidence, etc.) and its actual contents (118.40817°, 98.6°F, 80%, etc.). The *type* defines how the value field is interpreted: as a string, integer or floating point type, or as uninterpreted binary data (blobs).

The operator field not only allows attributes to contain data, but also to express simple constraints. There are two classes of operators: first, IS, the *actual* operator, is used to indicate a specific value. The second group includes binary comparisons (EQ, NE, LT, GT, LE, GE, corresponding to equality, inequality, less than, etc.) and "EQ_ANY" (which matches anything); these are collectively called *formal* operators.

Actuals are statements about data. So "latitude IS 33.9425, longitude IS 118.40817" might indicate a location, or "sensor IS seismic, value IS 7.0, confidence IS 80" might indicate a specific sensor reading.

Formals allow one to select sets of sensors, thus indicating which publish and subscribe operations should be connected. Thus, a subscription might indicate "latitude GT 33.5, latitude LT 34.0, sensor EQ seismic" to indicate seismic sensors in some area.

Formals and actuals can be mixed and used in publications, subscriptions, or filters.

The exact process of determining which publications and subscriptions are related is called matching. A *one-way match* compares all formal parameters of one attribute set against the actuals of the others (Figure 29.2). Any formal parameter that is missing a matching actual in the other attribute set causes the one-way match to fail (e.g. "confidence GT 0.5" must have an actual such as "confidence IS 0.7" and would not match "confidence IS 0.3," "confidence LT 0.7," or "confidence GT 0.7"). Two sets of attributes have a *complete match* if one-way matches succeed in both directions. In other words, attribute sets A and B match if the one-way match algorithm succeeds from both A to B and B to A.

Matching is used to associate publications and subscriptions and to activate filters as messages flow through the network.

Although matching is reasonably powerful, it does not perfectly cover all scenarios or tasks. Matching strikes a balance between ease of implementation and flexibility. For example, while attributes can easily define a square, they cannot directly operate on arbitrarily complex sensor detection regions. We expect applications to use attributes for rough matching and refine matching with application-specific code (such as with filters, Section 29.4).

For detailed examples of naming in diffusion, see the diffusion manual [4,6].

© 2005 by Chapman & Hall/CRC

```
one-way match:
given two attribute sets A and B
for each attribute a in A where a.op is a formal {
    matched = false
    for each attribute b in B where a.key = b.key and b.op is an actual
        if a.val compares with b.val using a.op, then matched = true
    if not matched then return false (no match)
}
return true (successful one-way match)
```

Figure 29.2. Our one-way matching algorithm.

29.3 Directed Diffusion Protocol Family

Publish/subscribe provides an application's view to a sensor network, and attribute-based naming is a detailed way to specify which sources and sinks communicate. The "glue" that binds the two are the directed diffusion algorithms for data dissemination. In a traditional network, communication is effected by routing, usually based on global addresses and routing metrics. Instead, we use the term data dissemination to emphasize the lack of global addresses, reliance on local rules, and, as described in Section 29.4, the use of application-specific in-network processing.

The original, two-phase directed diffusion uses several control messages to realize our publish/subscribe API: sinks send *interest* messages to find sources, sources use *exploratory data* messages to find sinks, and positive and negative *reinforcement* messages select or prune parts of the path. Early work (2) identified these primitives, described the concept of diffusion, and evaluated a specific algorithm that we now call *two-phase pull* diffusion. We found this algorithm ideal for some applications but, as our experience with sensor networks applications grew, we found two-phase pull a poor match for other classes of applications.

We see diffusion not as a single algorithm, but as a *family* of algorithms built from these primitives. Other algorithms provide better performance for some applications. We have recently made two additions to the diffusion protocol family: *one-phase push* and *one-phase pull* [7].

Another way to optimize diffusion performance is to use physical or application-specific information. The physical nature of a sensor network's deployment makes geographically scoped queries natural, prompting the development of geographically aided routing protocols such as geographic and energy-aware routing (GEAR) [8], greed permeter stateless routing (GPSR) [9], and rumor routing [10]. Application-specific information can also be exploited using filters (described in Section 29.4).

We expect application designers to match an appropriate algorithm with their application's requirements. Table 29.1 compares the interactions of the algorithms; we describe them below in more

Table 29.1. Comparison of interactions in diffusion algorithms. Asterisks indicate messages that are sent to all nodes (flooded or geographically scoped). All algorithms also have negative reinforcement messages

Protocol	Sink	Source
Two-phase pull	Interest* (every interest interval) Positive reinforcement (response to exp. Data)	Exploratory data* (every exploratory interval) Data (rate defined by app.)
One-phase pull	Interest* (every interest interval)	Data
Push	Positive reinforcement (response to exp. Data)	Exploratory data* (every exploratory interval) Data

© 2005 by Chapman & Hall/CRC

detail and review their performance in Section 29.5.4. More detail is the subject of current [2,7,11] and future research.

29.3.1 Two-Phase Pull Diffusion

The purpose of directed diffusion is to establish efficient *n*-way communication between one or more sources and sinks. Directed diffusion is a data-centric communication paradigm that is quite different from host-based communication in traditional networks. To describe the elements of diffusion, we take the simple example of a sensor network designed for tracking animals in a wilderness refuge.

Suppose that a user in this network would like to track the movement of animals in some remote subregion of the park. The user would subscribe to ''animal-track'' information, specified by a set of attributes. Sensors across the network publish animal-track information.

The user's application subscribes to data using a list of attribute–value pairs that describe a task using some task-specific naming scheme. Intuitively, attributes describe the data that are desired by specifying sensor types and possibly some geographic region. The user's node becomes a *sink*, creating an *interest* of attributes specifying a particular kind of data.

The interest is propagated from neighbor to neighbor towards sensor nodes in the specified region. A key feature of directed diffusion is that every sensor node can be *task-aware* — by this we mean that nodes store and interpret interests, rather than simply forwarding them along. In our example, each sensor node that receives an interest remembers which neighbor or neighbors sent it that interest. To each such neighbor, it sets up a *gradient*. A gradient represents both the direction towards which data matching an interest flows, and the status of that demand (whether it is active or inactive and possibly the desired update rate). After setting up a gradient, the sensor node redistributes the interest to its neighbors. When the node can infer where potential sources might be (e.g. from geographic information or existing similar gradients), the interest can be forwarded to a subset of neighbors. Otherwise, it will simply broadcast the interest to all of its neighbors.

Sensors indicate what data they may generate by publishing with an appropriate set of attributes. They thus become potential *sources*. As interests travel across the network, sensors with matching publications are triggered and the application activates its local sensors to begin collecting data. (Prior to activation we expect the node's sensors would be in a low-power mode.) The sensor node then generates *data* messages matching the interest. In directed diffusion, data are also represented using an attribute-based naming scheme.

Data is cached at intermediate nodes as it propagates toward sinks. Cached data are used for several purposes at different levels of diffusion. The core diffusion mechanism uses the cache to suppress duplicate messages and prevent loops, and it can be used to forward interests preferentially. (Since the filter core is primarily interested in an exact match, as an optimization, hashes of attributes can be computed and compared rather than complete data.) Cached data are also used for application-specific, in-network processing. For example, data from detections of a single object by different sensors may be merged to a single response based on sensor-specific criteria.

The initial data message from the source is marked as *exploratory* and is sent to all neighbors for which it has matching gradients. The initial flooding of the interest, together with the flooding of the exploratory data, constitutes the first phase of two-phase pull diffusion. If the sink has multiple neighbors, then it chooses to receive subsequent data messages for the same interest from a preferred neighbor (e.g. the one which delivered the first copy of the data message). To do this, the sink *reinforces* the preferred neighbor, which, in turn, reinforces its preferred upstream neighbor, and so on. The sink may also *negatively reinforce* its current preferred neighbor if another neighbor delivers better (lower latency) sensor data. This negative reinforcement propagates neighbor to neighbor, removing gradients and tearing down an existing path if it is no longer needed [2]. Negative reinforcements suppress loops or duplicate paths that may arise due to changes in network topology.

After the initial exploratory data message, subsequent messages are sent only on reinforced paths. (The path reinforcement, and the subsequent transmission of data along reinforced paths, constitutes

© 2005 by Chapman & Hall/CRC

the second phase of two-phase pull diffusion.) Periodically, the source sends additional exploratory data messages to adjust gradients in the case of network changes (due to node failure, energy depletion, or mobility), temporary network partitions, or to recover from lost exploratory messages. Recovery from data loss is currently left to the application. While simple applications with transient data (such as sensors that report their state periodically) need no additional recovery mechanism, we are also developing a retransmission scheme for applications that transfer large, persistent data objects [12].

This simplified description points out several key features of diffusion, and how it differs from traditional networking. First, diffusion is data-centric: all communication in a diffusion-based sensor network associates sources and sinks with interests and attribute-named data. Second, all communication in diffusion is neighbor–to–neighbor or hop–by–hop, unlike traditional data networks with end-to-end communication. Every node is an "end" in a sensor network. A corollary to this previous observation is that there are no "routers" in a sensor network. Each sensor node can interpret data and interest messages. This design choice is justified by the task specificity of sensor networks. Sensor networks are not general-purpose communication networks. Third, nodes do not need to have globally unique identifiers or globally unique addresses for regular operation. Nodes, however, do need to distinguish between neighbors. Fourth, because individual nodes can cache, aggregate and, more generally, process messages, it is possible to perform coordinated sensing close to the sensed phenomena. It is also possible to perform in-network data reduction, thereby resulting in significant energy savings. Finally, although our example describes a particular usage of the directed diffusion paradigm (a query–response type usage, see Figure 29.3), the paradigm itself is more general than that; we discuss several other usages next.

29.3.2 Push Diffusion

Two-phase pull diffusion works well for applications where a small number of sinks collect data from the sensor net, e.g. a user querying a network for detections of some tracked object. Another class of applications involves sensor-to-sensor communication within the sensornet. A simple example of this

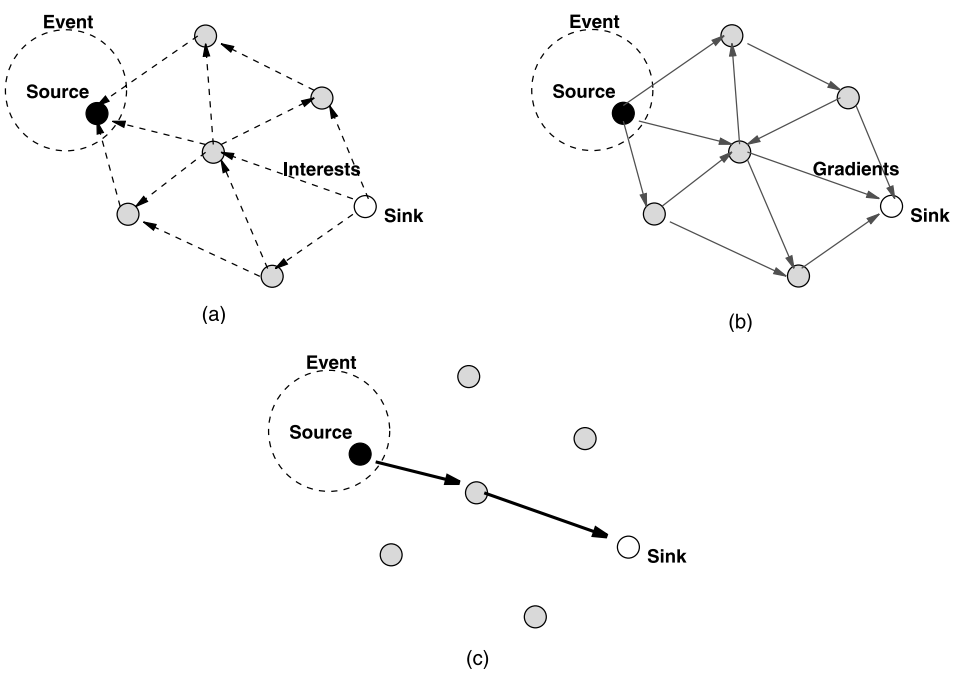


Figure 29.3. A simplified schematic for directed diffusion: (a) interest propagation; (b) initial gradients set-up; (c) data delivery along reinforced path.

© 2005 by Chapman & Hall/CRC

class of application might have sensors operating at a low duty cycle most of the time, but when one sensor detects something it triggers nearby sensors to become more active and vigilant. Push diffusion was motivated by applications such as these being developed at Sensoria Corporation, University of Wisconsin, and PARC. A characteristic of this class of application is that there are many sensors interested in data (activation triggers), and many that can publish such data, but the frequency of triggers actually being sent is fairly rare. Two-phase pull diffusion behaves poorly for this application, because all sensors actively send interests and maintain gradients to all other sensors even though nothing is detected.

One-phase push diffusion (or just push diffusion) was designed for this application. Although the API is the same as two-phase pull diffusion (except for a flag to indicate "push"), in the implementation, the roles of the source and sink are reversed. Sinks become passive, with interest information kept local to the node subscribing to data. Sources become active: exploratory data are sent throughout the network without interest-created gradients. As with two-phase pull, when exploratory data arrive at a sink a reinforcement message is generated and it recursively passes back to the source creating a reinforced gradient, and nonexploratory data follow only these reinforced gradients. Push can also take advantage of GEAR-style geographic optimizations.

Push is thus optimized for a different class of applications from two-phase pull: applications with many sources and sinks, but where sources produce data only occasionally. Push is not a good match for applications with many sources continuously generating data, since such data would be sent throughout the network even when not needed. Section 29.5.4.1 presents a performance comparison of push and two-phase pull diffusion for such an application.

29.3.3 One-Phase Pull Diffusion

A benefit of push diffusion compared with two-phase pull is that it has only one case where information is sent throughout the network (exploratory data) rather than two (interests and exploratory data). In large networks without geographically scoped queries, minimizing flooding can be a significant benefit. Inspired by efficiency of pull for some applications, we revisited two-phase pull to eliminate one of its phases of flooding.

One-phase pull is a subscriber-based system that avoids one of the two phases of flooding present in two-phase pull. As with two-phase pull, subscribers send interest messages that disseminate through the network, establishing gradients. Unlike two-phase pull, when an interest arrives at a source it does not mark its first data message as exploratory, but instead sends data only on the preferred gradient. The preferred gradient is determined by the neighbor who was the first to send the matching interest, thus suggesting the lowest latency path. Thus, one-phase pull does not require reinforcement messages, and the lowest latency path is implicitly reinforced.

One-phase pull has two disadvantages compared with two-phase pull. First, it assumes symmetric communication between nodes, since the data path (source to sink) is determined by lowest latency in the interest path (sink to source). Two-phase pull reduces the penalty of asymmetric communication, since the choice of data path is determined by lowest-latency exploratory messages, both in the source-to-sink direction. However, two-phase pull still requires some level of symmetry, since reinforcement messages travel reverse links. Although link asymmetry is a serious problem in wireless networks, many other protocols require link symmetry, including 802.11 and protocols that use link-level acknowledgments. As such, it is reasonable to assume that detecting and filtering such links will be done at the media access control (MAC) layer, allowing one-phase diffusion to work.

Second, one-phase pull requires interest messages to carry a flow-id. Although flow-id generation is relatively easy (uniqueness can be provided by MAC-level addresses or probabilistically with random assignment and periodic reassignment), this requirement makes interest size grow with the number of sinks. By comparison, though, with two-phase pull the number of interest messages grows in proportion to the number of sinks, so the cost here is lower. The use

© 2005 by Chapman & Hall/CRC

of end-to-end flow-ids also means that one-phase pull does not use only local information to make data dissemination decisions.

29.3.4 Using Geographic Cues to Limit Flooding

The physical nature of a sensor network's deployment makes geographically scoped queries natural. If nodes know their locations, then geographic queries can influence data dissemination, limiting the need for flooding to the relevant region.

GEAR extends diffusion when node locations and geographic queries are present [8]. GEAR is an extension to existing diffusion algorithms that replaces network-wide communication with geographically constrained communication. When added to one-phase or two-phase pull diffusion, GEAR's subscribers actively send interests into the network. However, queries expressing interest in a region are sent *towards* that region using greedy geographic routing (with support for routing around holes); flooding occurs only when interests reach the region rather than sent throughout the whole network. Exploratory data are sent only on gradients set up by interests, so the limited dissemination of interests also reduces the cost of exploratory data.

For one-phase push diffusion, GEAR uses the same mechanism to send exploratory data messages containing a destination region towards that region. This avoids flooding by allowing data senders to push their information only to subscribers within the desired region, which in turn will send reinforcements resulting in future data messages following a single path to the subscriber. In Section 29.4.2 we present a field experiment showing a performance comparison of push diffusion with and without GEAR using the PARC information-driven sensor querying (IDSQ) application.

We have also implemented GPSR [9] in the filter framework as an alternative to GEAR.

29.4 Facilitating In-Network Processing

Filters are our mechanism for allowing application-specific code to run in the network and assist diffusion and processing. Applications provide filters before deployment of a sensor network; or, in principle, filters could be distributed as mobile code packages at run time. Filters register what kinds of data they handle through matching; they are then triggered each time that kind of data enters the node. When invoked, a filter can arbitrarily manipulate the message, caching data, influencing how or where it is sent onward, or generating new messages in response. Uses of filters include routing, in-network aggregation, collaborative signal processing, caching, and similar tasks that benefit from control over data movement, as well as debugging and monitoring.

Filters use only one-way matching. A message entering a node triggers a filter if the attributes specified by the filter match the attributes in the message, but it does not require matching in the other direction. This approach allows filters to process data more generally with the publish/subscribe API.

The filter core is the system component responsible for interconnecting all hardware devices, applications, and filters. Even though, logically, messages pass from filter to filter, in practice all the messages pass through the filter core, which shepherds messages from filter to filter, according to filter priorities.

Priorities, defined at filter configuration, give a total ordering of all filters in a system. While message attributes select which filters can process a message, priorities specify the *order* in which those filters act.

Priorities are needed because the attributes of an incoming message may match multiple filters. In this case, filter priorities indicate which filter is invoked first. As described in Section 29.2, once a filter receives a message, it has total control over where the message will go next. A filter can pass the message to the next filter, modify the message and then send it, suppress it, generate messages in response of it, etc. Filters can also use the filter API to override the order of message processing by changing the priority field and/or messages attributes. Thus, a knowledgeable filter can direct a message anywhere in the diffusion stack. Since the contents or priority can change any time a message leaves a filter, all messages are always sent to the filter core, not immediately to the next filter.

© 2005 by Chapman & Hall/CRC

29.4.1 Implemented Filters

In this section we describe the set of filters that we have implemented and designed. As shown in Figure 29.4, the filter core interacts with all filters (rectangles), applications (circles at the top right), and radio hardware (the lozenge at the bottom). Solid and dashed rectangles represent existing and planned filters respectively. The core is responsible for dispatching all messages as they pass through the system and for suppressing duplicate messages.

Basic diffusion is implemented in the two-phase pull filter (labeled 'gradient' in Figure 29.4). This filter maintains gradients representing the state of any existing flows to all neighbors and is responsible for forwarding data messages using reinforced paths, in addition to periodically send out reinforcement messages and interests.

GEAR is a pair of filters that can optionally surround the two-phase pull filter to implement geographic and energy-aware routing [8]. Lacking prior information (such as geographic information or prior saved state), basic diffusion floods interests to all nodes in the network. GEAR overrides this behavior to forward messages with geographic assistance (interests are sent basically toward their geographic destination, but around any holes in the topology). GEAR consists of two filters, a *preprocessing* filter that sits above the two-phase pull filter to handle GEAR-specific beacon messages and to remove transient geographic information on arrival, and a *geographic routing* filter that acts after the two-phase pull filter to forward interests in a good direction.

Building on the GPSR definition [9], Greenstein at UCLA and Wang at USC have each implemented versions of GPSR. GPSR, like GEAR, uses geographic information to make informed neighbor selection when forwarding packets. One implementation was done as an extension of diffusion (as described above), another as a stand-alone routing module (independent of diffusion).

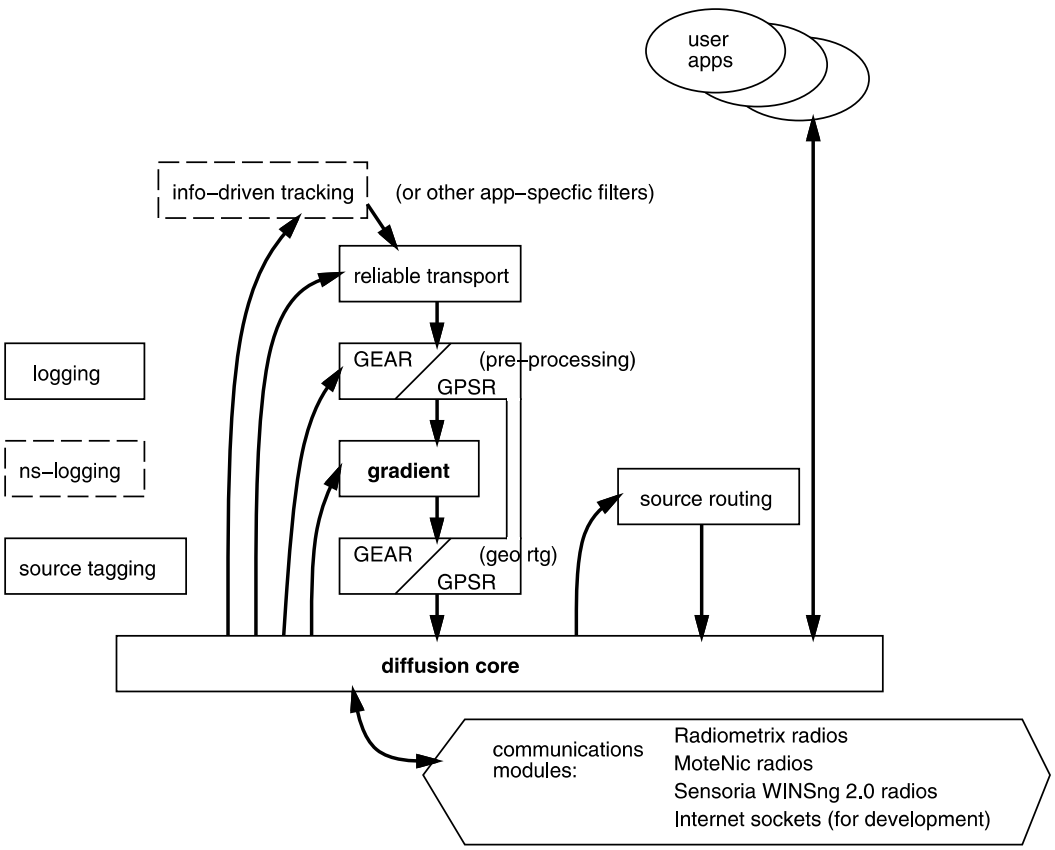


Figure 29.4. Current and planned filters in diffusion and how they interact.

© 2005 by Chapman & Hall/CRC

Reliable multi-segment transport (RMST) is a module that allows reliable transfers of large (multi-packet), uninterpreted data across unreliable links [12]. RMST is being used to investigate the trade-offs among MAC, transport, and application reliability. As a filter, it has two interesting characteristics. First, it caches data locally to support loss recovery, similar to approaches taken in reliable multicast [13] and SNOOP TCP[14], but at all hops rather than at the end-points or at base-stations only. Second, it implements a *back channel*, the reverse of the reinforced path created by the gradient filter. This back channel is used to propagate negative acknowledgment messages from the receiver to the sender.

The *information-driven tracking filter* is an example of how application-specific information can assist routing, proposed by researchers at Xerox PARC [15]. An important application of sensor networks is object tracking — multiple sensors may collaborate to identify one or more vehicles, estimating their position and velocity. Which sensors collaborate in this case is dependent on the direction of vehicle movement. They have proposed using current vehicle estimates (or ''belief state'') to involve the relevant sensors in this collaboration while allowing other sensors to remain inactive (conserving network bandwidth and battery power). While GEAR uses generic (geographic) information to reduce unnecessary communication, the information-driven tracking filter uses application-specific information to reduce communication further. As other applications are explored, we expect to develop other application-specific filters similar to the information-driving tracking filter.

One use of filters is logging information for debugging. We have implemented a *logging* filter for this purpose, and we are considering implementing an *ns-logging* filter for simulator-specific logging. These filters are shown to the left of the diffusion stack because they can be placed between any two modules.

Although this architecture was built to explore diffusion-style routing, for debugging purposes we also developed support for source routing. Source routing is provided as two filters: the *source tagging* filter functions similar to the logging filters, in that it can be configured anywhere in the diffusion stack. This filter adds a record of each node that the message passes through, much like the `traceroute` command used on the Internet. The *source routing* filter provides the opposite function. It takes a message that includes an attribute listing the path of nodes the message should take through the network and dispatches it along that path. A design principle of directed diffusion is local operation — nodes should not need to know information about neighbors multiple hops away. While source routing is directly opposite to this goal, it can be provided within our software framework, and is still sometimes a useful debugging tool.

29.5 Evaluation

In this section we evaluate diffusion using simulations and real-life experiments. We start by presenting simulation results showing the performance impact of diffusion. Then we describe nested queries, a new query model that reduces end-to-end traffic by doing application-level aggregation using diffusion's in-network processing capabilities. Later, we show examples where application performance is highly affected by choosing the best diffusion algorithm.

29.5.1 Implementation Experience

Several implementations of diffusion have existed in simulation and on several hardware platforms. Diffusion was first implemented by Intanagonwiwat *et al.* in simulation with ns-2 [2]. The first implementations for native hardware were for Linux/x86 (desktop computers) and WINSng 1.0 sensor nodes running Windows CE [Figure 29.5(c)]. More recent implementations added support for filters, PC/104 hardware with several kinds of radio [Figure 29.5(a)], and WINSng 2.0 nodes [Figure 29.5(d)] based on the SH-4 processor. The most recent release (3.2 as of this writing) includes nearly source-compatible support for the ns-2 simulator.

Researchers at UCLA have implemented *tiny diffusion*. Tiny diffusion is a simplified version of diffusion, which runs on the resource-constrained Mica motes [Figure 29.5(e)] running TinyOS [16]

© 2005 by Chapman & Hall/CRC

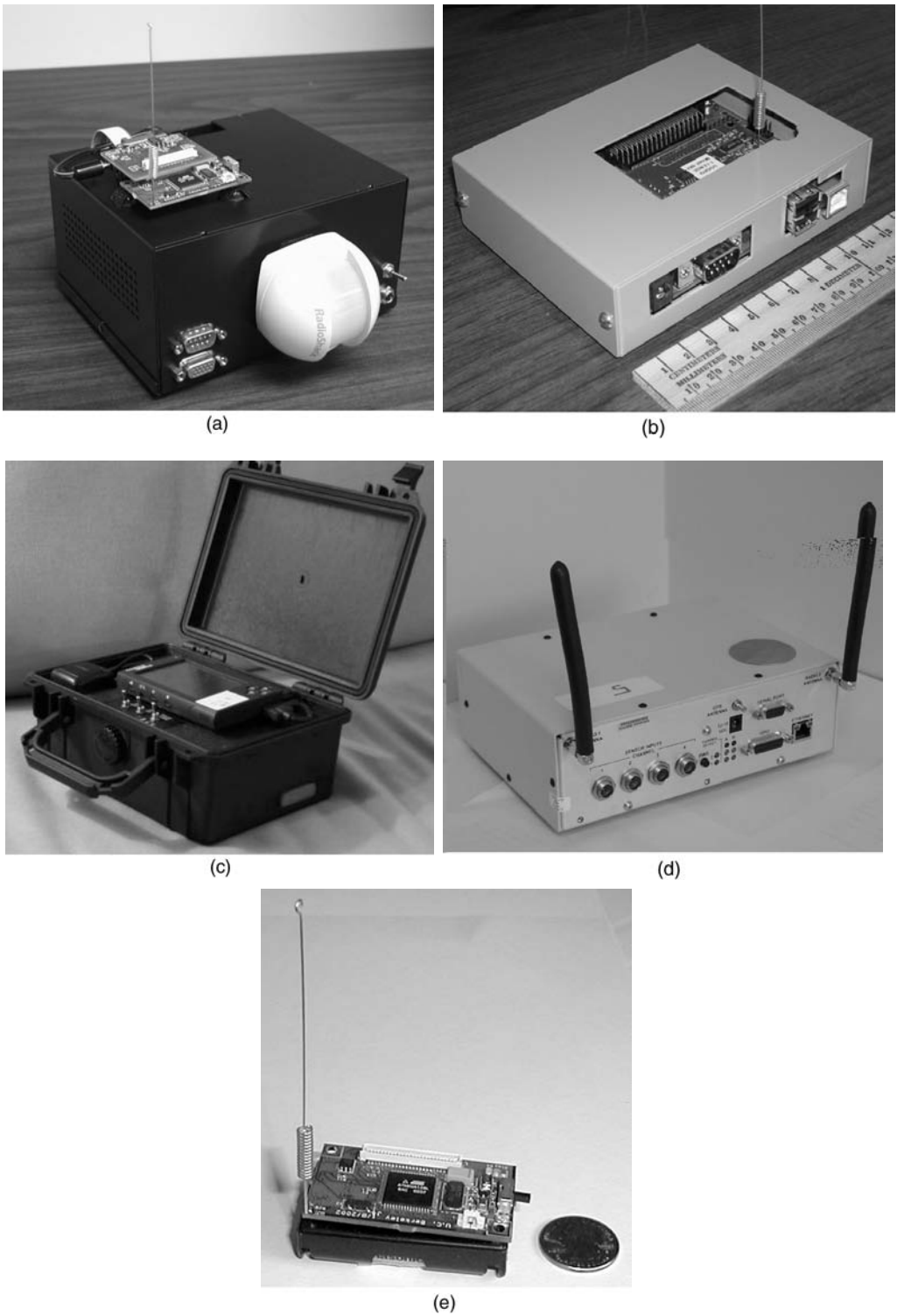


Figure 29.5. Diffusion hardware platforms. The mote only supports tiny diffusion: (a) Our PC/104 node; (b) an Intel Stayton node; (c) WINSng 1.0 node; (d) WINSng 2.0 node; (e) UCB Mica-1 mote.

© 2005 by Chapman & Hall/CRC

with a limited amount of memory and processing power. Although it does not include support for filters, this simplified version does support attributes, as well as a simplified version of the publish/subscribe API. Different versions of tiny diffusion have implemented both two-phase and one-phase pull diffusion.

29.5.2 Evaluation of Diffusion Design Choices

In this section we use packet-level simulation to explore, in some detail, the implications of some of our design choices. Such an examination complements and extends our description for the two-phase pull diffusion from Section 29.3.1. This section describes our methodology, compares the performance of diffusion against some idealized schemes, then considers the impact of network dynamics on simulation. Refer to Intanagonwiwat *et al.* [2] for a more detailed description of the simulations.

29.5.2.1 Goals, Metrics, and Methodology

We implemented a vehicle tracking instance of directed diffusion in the ns-2 [17] simulator (the current ns release with diffusion support can be downloaded from http://www.isi.edu/nsnam/ns). Our goals in conducting this evaluation study were: to verify and complement our analytic evaluation; to understand the impact of dynamics — such as node failures — on diffusion; and to study the sensitivity of directed diffusion performance to the choice of parameters.

We choose two metrics to analyze the performance of directed diffusion and to compare it with other schemes: mean dissipated energy and mean delay. Mean dissipated energy measures the ratio of total dissipated energy *per node* in the network to the number of *distinct* events seen by sinks. This metric computes the mean work done by a node in delivering useful tracking information to the sinks. The metric also indicates the overall lifetime of sensor nodes. Mean delay measures the mean one-way latency observed between transmitting an event and receiving it at each sink. This metric defines the temporal accuracy of the location estimates delivered by the sensor network. We study these metrics as a function of sensor network size.

In order to study the performance of diffusion as a function of network size, we generate a variety of sensor fields of different sizes. In each of our experiments, we study five different sensor fields, ranging from 50 to 250 nodes in increments of 50 nodes. Our 50 node sensor field is generated by randomly placing the nodes in a 160 m by 160 m square. Each node has a radio range of 40 m. Other sizes are generated by scaling the square and keeping the radio range constant in order to keep the average density of sensor nodes approximately constant. We do this because the macroscopic connectivity of a sensor field is a function of the average density. If we had kept the sensor field area constant but increased network size, then we might have observed performance effects not only due to the larger number of nodes but also due to increased connectivity. Our methodology factors out the latter, allowing us to study the impact of network size alone on some of our mechanisms.

The ns-2 simulator implements a 1.6 Mb/s 802.11 MAC layer. Our simulations use a modified 802.11 MAC layer. To mimic realistic sensor network radios more closely [18], we altered the ns-2 radio energy model such that the idle-time power dissipation was about 35 mW, or nearly 10% of its receive power dissipation (395 mW), and about 5% of its transmit power dissipation (660 mW). This MAC layer is not completely satisfactory, since energy efficiency provides a compelling reason for selecting a time-division multiple access (TDMA)-style MAC for sensor networks rather than one using contention-based protocols [1]. Briefly, these reasons have to do with energy consumed by the radio during idle intervals; with a TDMA-style MAC, it is possible to put the radio in standby mode during such intervals. By contrast, an 802.11 radio consumes as much power when it is idle as when it receives transmissions. In Section 29.5.2.3 we analyze the impact of a MAC energy model in which listening for transmissions dissipates as much energy as receiving them.

Finally, data points in each graph represent the mean of ten scenarios with 95% confidence intervals. Refer to Intanagonwiwat *et al.* [2] for a more detailed description of the methodology used.

© 2005 by Chapman & Hall/CRC

29.5.2.2 Comparing Diffusion with Alternatives

Our first experiment compares diffusion with an omniscient multicast and a flooding scheme for data dissemination in networks. Figure 29.6(a) shows the average dissipated energy per packet as a function of network size. Omniscient multicast dissipates a little less than a half as much energy per packet per node than flooding. It achieves such energy efficiency by delivering events along a single path from each source to every sink. Directed diffusion has noticeably better energy efficiency than omniscient multicast. For some sensor fields, its dissipated energy is only 60% that of omniscient multicast. As with omniscient multicast, it also achieves significant energy savings by reducing the number of paths over which redundant data are delivered. In addition, diffusion benefits significantly from *in-network aggregation*. In our experiments, the sources deliver identical location estimates, and intermediate nodes *suppress* duplicate location estimates. This corresponds to the situation where there is, for example, a single vehicle in the specified region.

Figure 29.6(b) plots the average delay observed as a function of network size. Directed diffusion has a delay comparable to omniscient multicast. This is encouraging. To a first approximation, in an uncongested sensor network and in the absence of obstructions, the shortest path is also the lowest delay path. Thus, our reinforcement rules seem to be finding the low delay paths. However, the delay experienced by flooding is almost an order of magnitude higher than other schemes. This is an artifact of the MAC layer: to avoid broadcast collisions, a randomly chosen delay is imposed on all MAC broadcasts. Flooding uses MAC broadcasts exclusively. Diffusion only uses such broadcasts to propagate the initial interests. On a sensor radio that employs a TDMA MAC-layer we might expect flooding to exhibit a delay comparable to the other schemes.

In summary, directed diffusion exhibits better energy dissipation than omniscient multicast and has good latency properties.

29.5.2.3 Effects of Data Aggregation

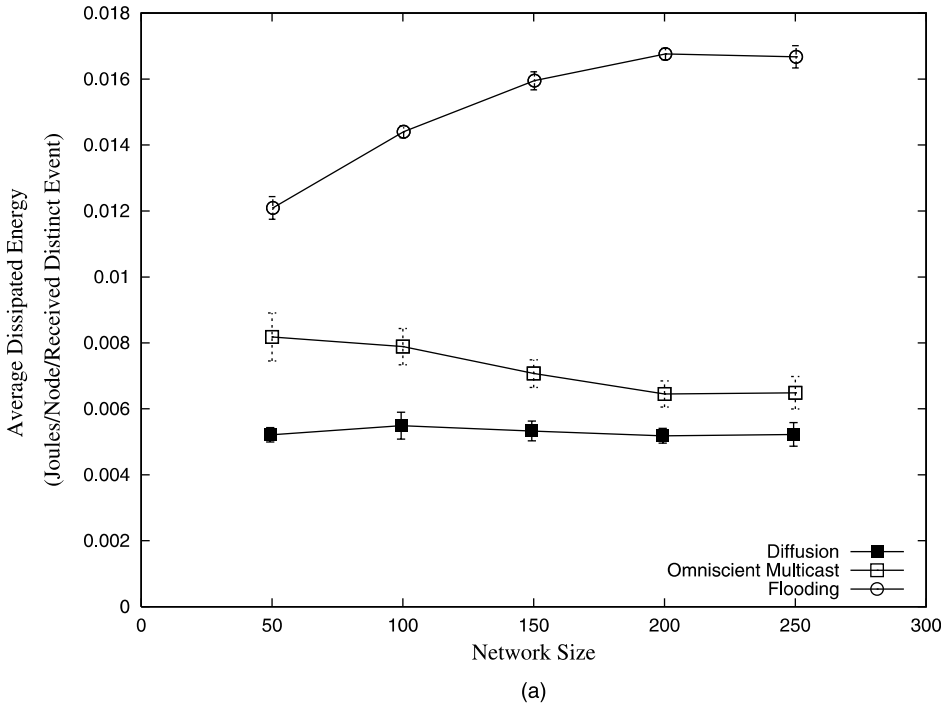
To explain what contributes to directed diffusion's energy efficiency, we now describe two separate experiments. In both of these experiments we do not simulate node failures. First, we compute the energy efficiency of diffusion with and without aggregation. Recall from Section 29.5.2.2 that, in our simulations, we implement a simple aggregation strategy in which a node suppresses identical data sent by different sources. As Figure 29.7(a) shows, diffusion expends nearly five times as much energy, in smaller sensor fields, as when it can suppress duplicates. In larger sensor fields, the ratio is 3. Our conservative negative reinforcement rule accounts for the difference in the performance of diffusion without suppression as a function of network size. With the same number of sources and sinks, the larger network has longer alternate paths. These alternate paths are truncated by negative reinforcement because they consistently deliver events with higher latency. As a result, the larger network expends less energy without suppression. We believe that suppression also exhibits the same behavior, but the energy difference is relatively small.

29.5.2.4 Effects of Radio Energy Model

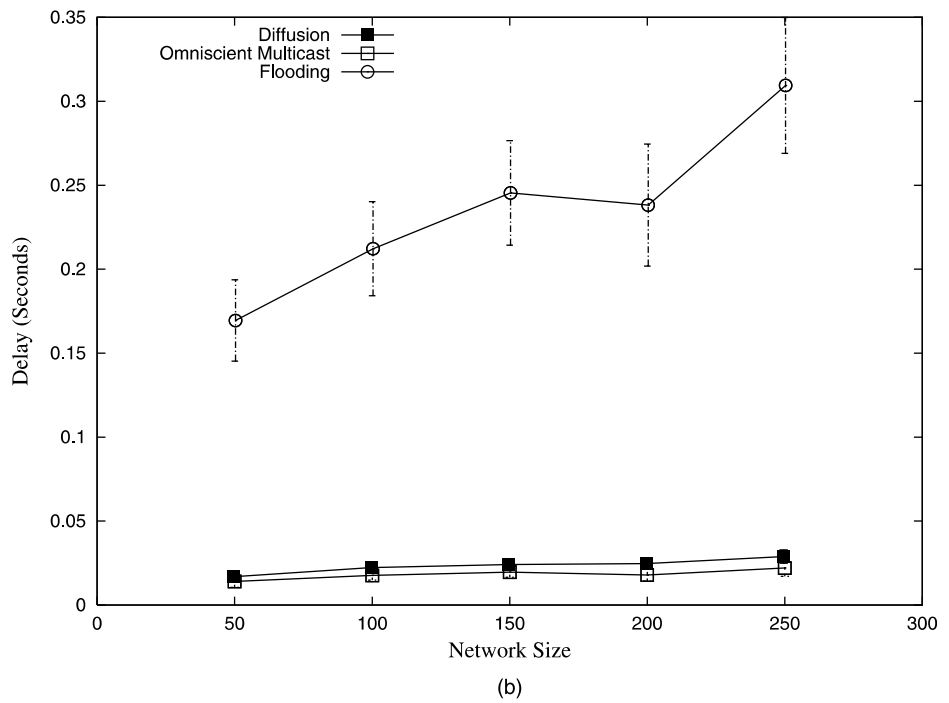
Finally, we evaluate the sensitivity of our comparisons (Section 29.5.2.2) to our choice of energy model. Sensitivity of diffusion to other factors (numbers of sinks, size of source region) is discussed in greater detail by Intanagonwiwat *et al.* [19].

In our comparisons, we selected radio power dissipation parameters to more closely mimic realistic sensor radios [18]. We re-ran the comparisons of Section 29.5.2.2, but with power dissipation comparable to the AT&T Wavelan: 1.6 W transmission, 1.2 W reception and 1.15 W idle [20]. In this case, as Figure 29.7(b) shows, the distinction between the schemes disappears. In this regime, we are better off flooding all events. This is because idle time energy utilization completely dominates the performance of all schemes. This is the reason why sensor radios try very hard to minimize listening for transmissions.

© 2005 by Chapman & Hall/CRC



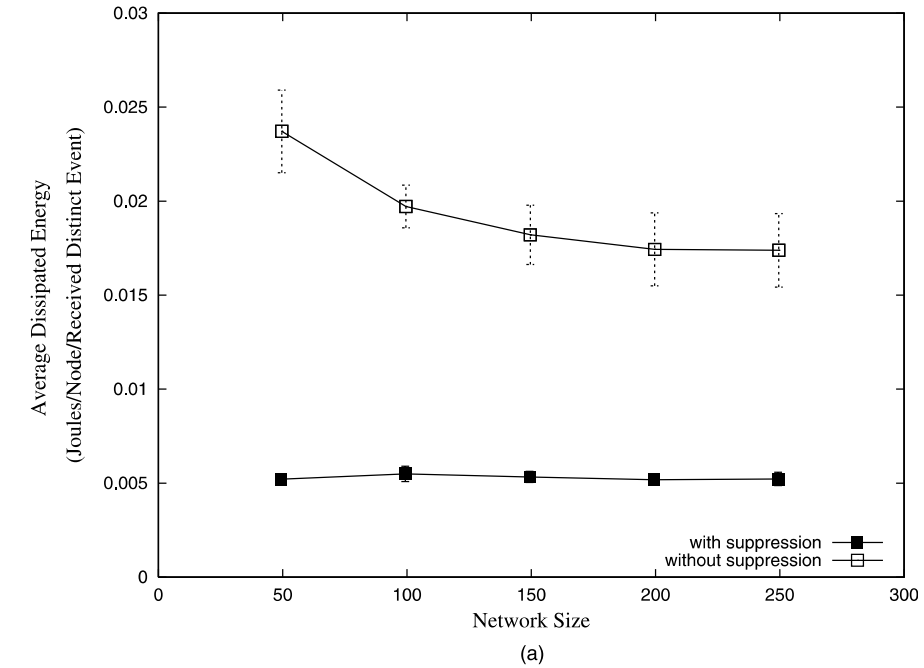
(a)



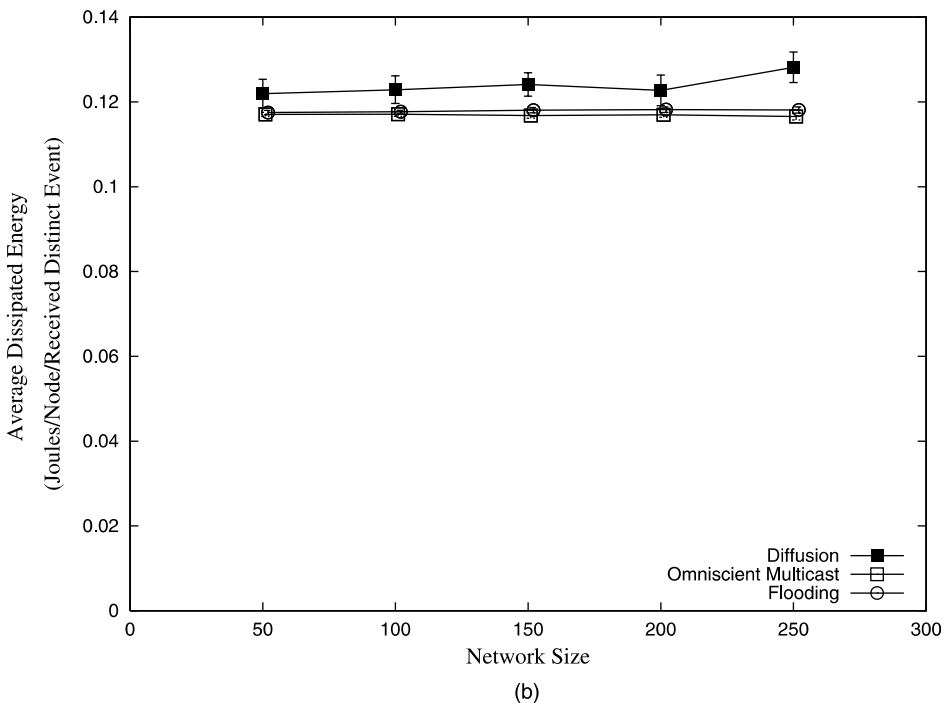
(b)

Figure 29.6. Directed diffusion compared with flooding and omniscient multicast: (a) mean dissipated energy; (b) mean delay.

© 2005 by Chapman & Hall/CRC



(a)



(b)

Figure 29.7. Impact of various factors on directed diffusion: (a) duplicate suppression; (b) high idle radio power.

© 2005 by Chapman & Hall/CRC

29.5.3 Evaluation of In-Network Processing

Real-world events often occur in response to some environmental change. For example, a person entering a room is often correlated with changes in light or motion, or a flower's opening is correlated with the presence or absence of sunlight. Multi-modal sensor networks can use these correlations by triggering a secondary sensor based on the status of another, in effect nesting one query inside another. Reducing the duty cycle of some sensors can reduce the overall energy consumption (if the secondary sensor consumes more energy than the initial sensor, e.g. as an accelerometer triggering a global positioning system receiver) and network traffic (e.g. a triggered imager generates much less traffic than a constant video stream). Alternatively, in-network processing might choose the best application of a sparse resource (e.g. a motion sensor triggering a steerable camera).

Figure 29.8 shows two approaches for a user to cause one sensor to trigger another in a network. In both cases we assume that sensors know their locations and not all nodes can communicate directly. Part (a) shows a direct way to implement this: the user queries the initial sensors (small squares); when a sensor is triggered, the user queries the triggered sensor (the small gray circle). The alternative, shown in part (b), is a nested, two-level approach where the user queries the triggered sensor, which then subtasks the initial sensors. This nested query approach grew out of discussions with Philippe Bonnet and embedded database query optimization in his COUGAR database [21].

The advantage of a nested query is that data from the initial sensors can be interpreted directly by the triggered sensor, rather than passing through the user. In monitoring applications, the initial and triggered sensors would often be quite close to each other (to cover the same physical area), while the user would be relatively distant. A nested query localizes data traffic near the triggering event rather than sending it to the distant user, thus reducing network traffic and latency. Since energy-conserving networks are typically low-bandwidth and may be higher latency, reduction in latency can be substantial, and reductions in aggregate bandwidth to the user can mean the difference between an overloaded and operational network. The challenges for nested queries are how to match the initial and triggered sensors robustly and how to select a good triggered sensor if only one is desired.

Implementation of direct queries is straightforward with attribute-addressed sensors. The user subscribes to data for initial sensors and when something is detected the user requests the status of the triggered sensor (either by subscribing or asking for recent data). Direct queries illustrate the utility of

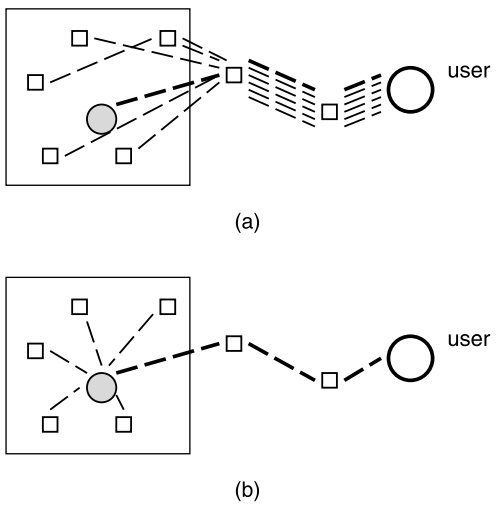


Figure 29.8. Two approaches to implementing nested queries. Squares are initial sensors, gray circles are tiggered sensors, and the large circle is the user. Thin dashed lines represent communication to initial sensors; bold lines are communication to the triggered sensor.

© 2005 by Chapman & Hall/CRC

predefined attributes identifying sensor types. Diffusion may also make use of geography to optimize routing.

Nested queries can be implemented by enabling code at each triggered sensor that watches for a nested query. This code then subtasks the relevant initial sensors and activates its local triggered sensor on demand. If multiple triggered sensors are acceptable but there is a reasonable definition of which one is best (perhaps, the most central one), then it can be selected through an election algorithm. One such algorithm would have the triggered sensors nominate themselves after a random delay as the "best," informing their peers of their location and election (this approach is inspired by SRM repair timers [22]). Better peers can then dispute the claim. Use of location as an external frame of reference defines a best node and allows timers to be weighted by distance to minimize the number of disputed claims.

In the next section we evaluate nested queries with experiments in our testbed.

29.5.3.1 Goals and Methodology

To validate our claim about the potential performance benefits of this implementation, we measured the performance of an application that uses nested queries against one that does not.

The application is similar to that described in Figure 29.8: a user requests acoustic data correlated with (triggered by) light sensors. For this experiment, we used our testbed of 14 PC/104 sensor nodes distributed on two floors of ISI (Figure 29.9). These sensors are connected by Radiometrix RPC modems (off-the-shelf, 418 MHz, packet-based radios that provide about 13 kb/s throughput) with 10 dB attenuators on the antennas to allow multi-hop communications in our relatively confined space. The exact topology varies depending on the level of radio-frequency activity, and the network is typically five hops across. We placed the user "U" at node 39, the audio sensor "A" at node 20, and light sensors "L" at nodes 16, 25, 22, and 13. It is one hop from the light sensors to the audio sensor, and two hops from there to the user node. To provide a reproducible experiment we simulate light data to change automatically every minute on the minute. Light sensors report their state every 2 s (no special attempt is made to synchronize or desynchronize sensors). Audio sensors generate simulated audio data each time any light sensor changes state. Light and audio data messages are about 100 bytes long.

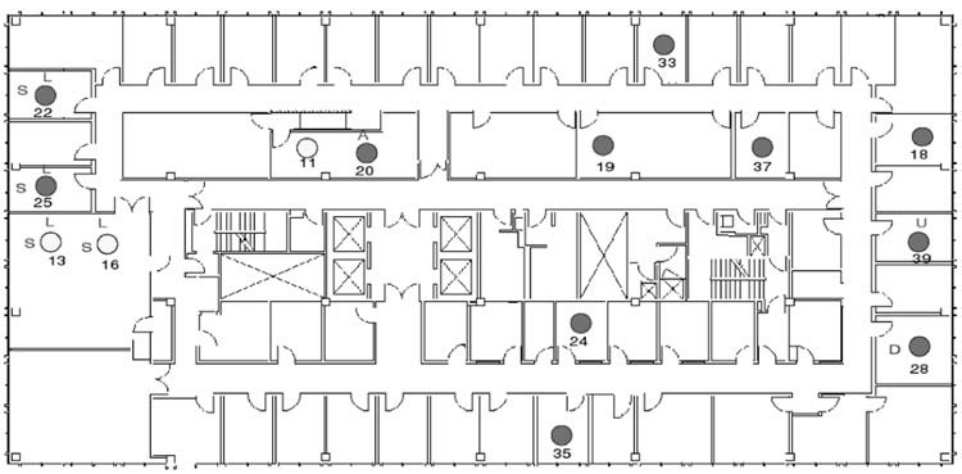


Figure 29.9. Node positions in our sensor testbed. Light nodes(11,13,16) are on the 10th floor; the remaining dark nodes are on the 11th floor. Radio range varies greatly depending on node position, but the longest stable link was between nodes 20 and 25.

© 2005 by Chapman & Hall/CRC

29.5.3.2 Nested Queries Benefits

Figure 29.10 shows the percentage of light-change events that successfully result in audio data delivered to the user. (Data points represent the mean of three 20-min experiments and show 95% confidence intervals.) The total number of possible events is the number of times all light sources change state, and a successful event is audio data delivered to the user. These delivery rates do not reflect per-hop message delivery rates (which are much higher), but rather the cumulative effect of sending best-effort data across three or five hops for nested or flat queries respectively.

This system is very congested and the network exhibits very high loss rates. Our current MAC is quite unsophisticated, performing only simple carrier detection and lacking RTS/CTS or ARQ. Since all messages are broken into several 27-byte fragments, loss of a single fragment results in loss of the whole message, and hidden terminals are endemic to our multi-hop topology, this MAC performs particularly poorly at high load. Missing events translate into increased detection latency. Although a sensor network could afford to miss a few events (since they would be retransmitted the next time the sensor is measured), these loss rates are unacceptably high for an operational system.

However, this experiment sharply contrasts the bandwidth requirements of nested and flat queries. Even with one sensor the flat query shows significantly greater loss than the nested query, because both light and audio data must travel to the user. Both flat and nested queries suffer greater loss when more sensors are present, but the one-level query falls off further. Comparing the delivery rates of nested queries with one-level queries shows that localizing the data to the sensors is very important to parsimonious use of bandwidth. In an uncongested network we expect that nested queries would allow operation with a lower level of data traffic than one-level queries and so would allow a lower radio duty cycle and a longer network lifetime.

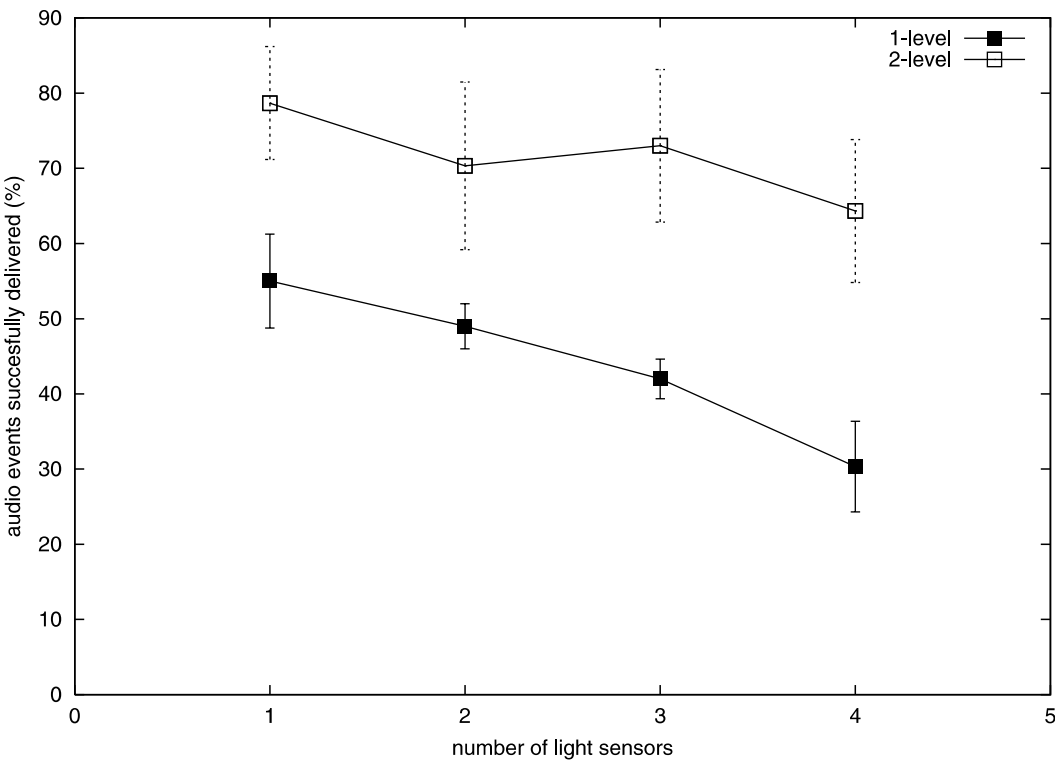


Figure 29.10. Percentage of audio events successfully delivered to the user.

© 2005 by Chapman & Hall/CRC

29.5.4 Application Performance with Different Diffusion Algorithms

In Section 29.3 we described a series of diffusion algorithms that were designed in response to application needs. This section describes two applications developed or inspired by other researchers that benefit from push and GEAR, and it quantifies the performance gains in switching diffusion algorithms.

29.5.4.1 One-Phase Push versus Two-Phase Pull Diffusion

Our first application considers trade-offs in push against two-phase pull versions of diffusion. In two-phase pull, data sinks are active, sending out interests, while sources are passive until interests arrive. By contrast, with push, the data sources are active, sending out data when it arrives. Push is designed for the case when there are many active sinks (listening for data), but relatively few nodes actually generating data. A common case of this kind of application is where many nodes are *cross-subscribed* to each other but mostly quiescent, all waiting for a triggering event to happen.

We explored this kind of application in the BAE sensor network testbed composed of 15 Sensoria WINSng 2.0 nodes. (These are 32-bit embedded computers with megabytes of memory and two independent, frequency-hopping radios that send data at about 20 kb/s.) The application was inspired by applications at the University of Wisconsin and PARC that employ cross-subscription. However, because those applications were not available to us at the time, we implemented a comparable application with a field of seven sensor nodes, all cross-subscribed to each other. When any one sensor changes state, all sensors send their readings to a triggered node that aggregates these readings and sends the aggregated result to the user. To control traffic, sensors were set to generate readings every 5 s and to change state every minute.

Figure 29.11 shows a trace of communication rates across this experiment, where each point represents the number of packets sent over the last 30 s. Two things stand out about this graph.

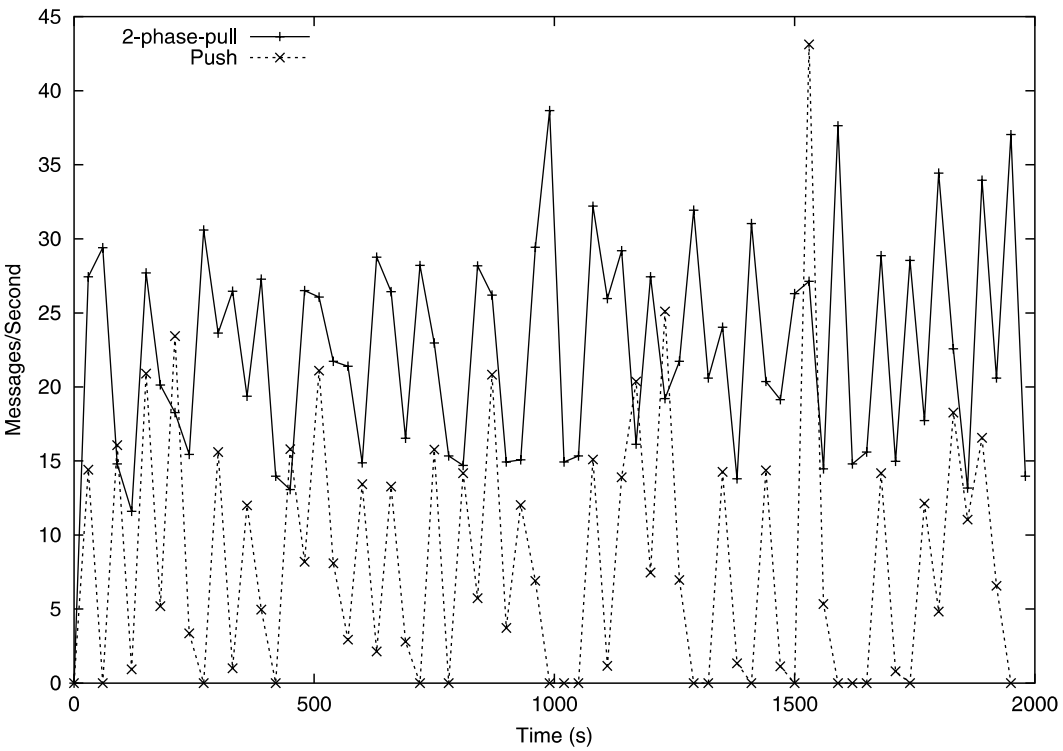


Figure 29.11. Push versus two-phase pull diffusion with a cross-subscription application.

© 2005 by Chapman & Hall/CRC

First, the application's traffic is quite bursty. Second, push (the dotted line) is consistently able to outperform two-phase pull (the solid line), transferring the same data with about 60% fewer messages.

Part of the saving in this experiment is because push is better suited to this application than two-phase pull. With many nodes cross-subscribed to each other, each will be frequently sending out interested messages to the network. With push, these interests are not sent; the only flooded messages are exploratory data.

If the sensors pushed relatively few detection events, then benefits of push would be greater still. In this case, data are sent every 5 s from each sensor to the others and so sensors are not quiescent.

29.5.4.2 Geographic Constraints

Researchers at Xerox PARC have suggested IDSQ [23], an information-theoretic approach to sensornet tracking. With their approach, one node (the leader) keeps track of the current target estimate. It periodically computes which other sensor can add the most information about the target location and then transfers leadership to that node through a process called *state transfer*. To keep the system state consistent, leader election includes a suppression process where a leader informs other nodes not to become active, additional leaders themselves. Suppression messages are sent when the target is first detected and as it moves through the network. State-transfer messages occur twice each second.

This application should benefit from push in the same way as the previous application (Section 29.5.4.1). In addition, suppression and state transfer are both geographically scoped actions. To investigate the benefits of geographically scoped communications jointly with them, we evaluated this application both with and without GEAR [8]. This application runs over 18 WINSng 2.0 nodes in the PARC sensor network testbed. Sensor data in this case are generated by one or two humans pulling a cart with prerecorded acoustic data mimicking a large vehicle. The first simulated vehicle starts at 120 s, and the second at about 170 s.

Figure 29.12 shows the message rates for this application. As can be seen, geographic scoping reduces message counts by 40%. This reduction is due to scoping of suppression messages. State-transfer messages in this application are sent to a single point and so are also geographically directed; however, this early implementation of push with GEAR did not support constraint of messages to a single point, only to regions, and so state-transfer messages were flooded. We would expect a larger reduction in control overhead now that push with GEAR constrains control traffic directed to a point.

29.5.4.3 Discussion

These case studies illustrate the importance of matching the application to an appropriate data dissemination algorithm. They also illustrate the complexity of selecting the best algorithm for a given application. Application designers are experts in their field, not networking, and so do not always have the best perspective to chose between several similar algorithms. The effects of selecting a diffusion algorithm can easily be masked by application errors. Our comparison of algorithms below is a first step to provide guidance to application designers, but an important area of future work is tools to help visualize and debug communication patterns in distributed, sensor-network applications.

To some extent it is a misstatement to suggest that there is a best algorithm for a single application. A sophisticated application like IDSQ has different patterns of communication in different parts of the application, and so requires *different* diffusion algorithms for different parts of the application. This supports our claim that a range of general and application-specific communication protocols are required for efficient data dissemination in sensor networks, both for different applications and even in a single application.

A more specific result of these field studies concerns the appropriate means to select between algorithms. We had originally assumed that diffusion could infer the correct algorithm from the user's commands. For example, if geographic information was present, then GEAR optimizations would be used. This approach proved too fragile for several reasons. First, it is prone to error. A misconfigured set

© 2005 by Chapman & Hall/CRC

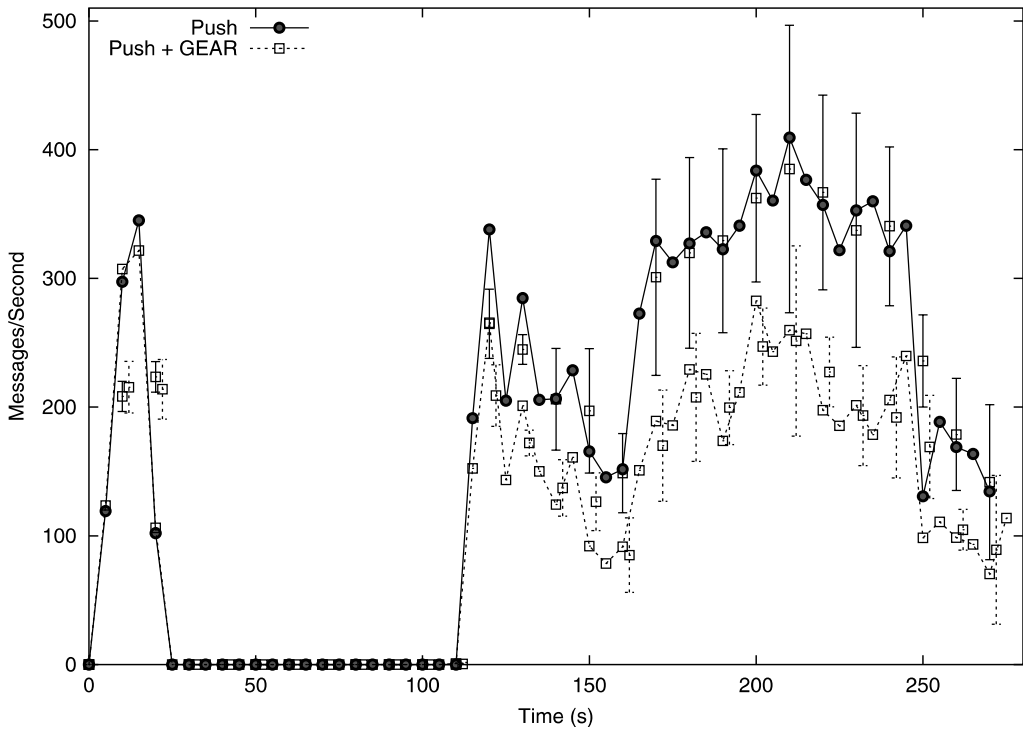


Figure 29.12. Push diffusion with and without GEAR over the IDSQ application.

of attributes can be syntactically correct but will not select the intended algorithm. The application will still run, but at greatly reduced performance. This problem is quite difficult to identify and correct, because performance of a distributed system can be difficult to measure, poor performance can be due to many causes, and the difference between correct code and incorrect is subtle. Second, as the numbers of alternative algorithms grow, it is no longer possible to distinguish between them automatically. The choice between algorithms often depends on characteristics of the application known only to the programmer, such as the communications patterns. A self-tuning system would be ideal, but collecting information for tuning requires communication itself, and so will add its own overhead. For these reasons we now select algorithms explicitly as an attribute to publish and subscribe calls. We view the algorithm attribute as a programmer-provided assertion, much as annotations are used in distributed-shared-memory systems (e.g. Munin [24]).

29.6 Related Work

Space constraints preclude a detailed summary of related work; for a more detailed study, see the related work sections of prior papers [2,6,7,11].

Our publish/subscribe API was designed with researchers from MIT's Lincoln Labs [3]. The approach was inspired by prior, Internet-based publish/subscribe systems (examples include [25–28]). The concept of formals and actuals in matching is derived from Linda [29], and our application of attribute-based naming to sensor networks was inspired by SRM [22].

The diffusion approach to data dissemination can be compared with *ad hoc* routing protocols (Broch *et al.* [30] survey several protocols, including DSR and AODV). Unlike end-to-end Internet protocols, diffusion encourages in-network processing. In-network processing in diffusion is similar to active networking [31], although the domain is quite different. DataSpace provided geographic routing [32], but not tied with attributes.

© 2005 by Chapman & Hall/CRC

In sensor networks, Piconet provided a fairly static system with devices, concentrators, and hosts [33]. SPIN evaluates several variants of flooding for wireless sensor networks [34], and INS designed a naming system for Internet-based hosts [35]. Neither exploited in-network processing. COUGAR adopted a database-like approach to sensor networks [21] and inspired our approach to nested queries.

29.7 Conclusion

We have described directed diffusion, a data-centric approach to information dissemination for sensor networks. Building on a publish/subscribe API and attribute-based naming, the diffusion primitives support a family of routing algorithms optimized for different applications. Filters support in-network processing to allow applications to manipulate data as it flows through the network.

Acknowledgments

Directed diffusion builds on the work of many people. Van Jacobson suggested the concept of diffusion as a communication strategy. Chalermek Intanagonwiwat designed and evaluated the basic algorithm. Dan Coffin collaborated in definition of the publish/subscribe API. Philippe Bonnet inspired our approach to nested queries. Yan Yu, Fred Stann, Ben Greenstein, and Xi Wang developed filters. Philippe Bonnet and Joe Reynolds were early users of diffusion, Jim Reich and Julia Liu early users of push, and Eric Osterweil of one-phase pull diffusion. The following are reprinted with permission of ACM: Figures 29.1, 29.3, and 29.8–29.10 are taken from [6]; Figures 29.11 and 29.12 and Table 29.1 are taken from [7].

References

[1] Pottie, G.J. and Kaiser, W.J., Embedding the internet: wireless integrated network sensors, *Communications of the ACM*, 43(5), 51, 2000.

[2] Intanagonwiwat, C. *et al.*, Directed diffusion: a scalable and robust communication paradigm for sensor networks, in *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, Boston, MA, USA, August 2000, ACM, 56.

[3] Coffin, D.A. *et al.*, Declarative *ad-hoc* sensor networking, in *Proceedings of the SPIE Integrated Command Environments Conference*, San Diego, CA, USA, July 2000, SPIE.

[4] Silva, F. *et al.*, *Network Routing Application Programmer's Interface (API) and Walk Through 9.0.1*, USC/Information Sciences Institute, December 2002.

[5] Braden, R. *et al.*, From protocol stack to protocol heap–role-based architecture, in *Proceedings of the ACM Workshop on Hot Topics in Networks I*, Princeton, NJ, USA, October 2002, ACM, 17, <http://www.isi.edu/newarch/DOCUMENTS/notrba.paper.pdf>.

[6] Heidemann, J. *et al.*, Building efficient wireless sensor networks with low-level naming, in *Proceedings of the Symposium on Operating Systems Principles*, Chateau Lake Louise, Banff, Alberta, Canada, October 2001, ACM, 146.

[7] Heidemann, J. *et al.*, Matching data dissemination algorithms to application requirements, in *Proceedings of the ACM SenSys Conference,* Los Angeles, CA, USA, November 2003, ACM, 218, <http://www.isi.edu/~johnh/PAPERS/heidemann03b.html>.

[8] Yu, Y. *et al.*, Geographical and energy aware routing: a recursive data dissemination protocol for wireless sensor networks, Technical Report TR-01-0023, University of California, Los Angeles, Computer Science Department, 2001.

[9] Karp, B. and Kung, H.T., GPSR: greedy perimeter stateless routing for wireless networks, in *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, Boston, MA, USA, August 2000, ACM, 243.

© 2005 by Chapman & Hall/CRC

[10] Braginsky, D. and Estrin, D., Rumor routing algorithm for sensor networks, in *Proceedings of the First ACM Workshop on Sensor Networks and Applications*, Atlanta, GA, USA, October 2002, ACM, 22.

[11] Intanagonwiwat, C. *et al.*, Directed diffusion for wireless sensor networking, *ACM/IEEE Transactions on Networking*, 11(1), 2, 2003.

[12] Stann, F. and Heidemann, J., Rmst: reliable data transport in sensor networks, in *Proceedings of the First International Workshop on Sensor Net Protocols and Applications*, Anchorage, AK, USA, April 2003, USC/Information Sciences Institute, IEEE, 102.

[13] Floyd, S. *et al.*, A reliable multicast framework for light-weight sessions and application level framing, in *Proceedings of the ACM SIGCOMM Conference*, Cambridge, MA, August 1995, ACM, 342.

[14] Balakrishnan, H. *et al.*, Improving TCP/IP performance over wireless networks, in *Proceedings of the First ACM Conference on Mobile Computing and Networking*, Berkeley, CA, USA, November 1995, ACM, 2.

[15] Zhao, F. *et al.*, Information-driven dynamic sensor collaboration for tracking applications, *IEEE Signal Processing Magazine*, 19(2), 61, 2002, <http://www.parc.xerox.com/spl/projects/consense/pub/ieee_spm-pdf>.

[16] Hill, J. *et al.*, System architecture directions for network sensors, in *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, Cambridge, MA, USA, November 2000, ACM, 93.

[17] Heidemann, J. *et al.*, Advances in network stimulation, *IEEE computer*, 33(5), 59, 2000, (an expanded version is available as USC CSD TR 99–702b), <hhttp://www.isi.edu/~johnh/PAPERS/Breshlan00a.html>.

[18] Kaiser,W.J., WINS NG 1.0 Transceiver Power Dissipation Specifications, Sensoria Corp.

[19] Intanagonwiwat, C. *et al.*, Directed diffusion: a scalable and robust communication paradigm for sensor networks, Technical Report 00-732, University of Southern California, March 2000.

[20] Stemm, M. and Katz, R.H. Measuring and reducing energy consumption of network interfaces in hand-held devices, *IEICE Transactions on Communications*, E80-B(8), 1125, 1997.

[21] Bonnet, P., *et al.*, Query processing in a device database system, Technical Report TR99-1775, Cornell University, October 1999.

[22] Floyd, S. and Jacobson, V., Link-sharing and resource management models for packet networks, *ACM/IEEE Transactions on Networking*, 3(4), 365, 1995.

[23] Chu, M. *et al.*, Scalable information-dirven sensor querying and routing for *ad hoc* hetereogeneous sensor networks, Technical Report P2001-10113, XEROX Palo Alto Research Center, May 2001.

[24] Carter, J.B., Implementation and performance of Munin, in *Proceedings of the Thirteenth Symposium on Operating Systems Principles*, October 1991, ACM, 152.

[25] Peterson, L.L., A Yellow-Pages service for a local-area network, in *Proceedings of the ACM SIGCOMM Conference '87*, August 1987, 235.

[26] Birman, K.P., The process group approach to reliable distributed computing, *Communications of the ACM*, 36(12), 36, 1993.

[27] Oki, B. *et al.*, The information bus — an architecture for extensible distributed systems, in *Proceedings of the 14th Symposium on Operating Systems Principles*, Asheville, NC, USA, December 1993, ACM, 58.

[28] Carzaniga, A. *et al.*, Design and evaluation of a wide-area event notification service, *ACM Transactions on Computer Systems*, 19(3), 332, 2001.

[29] Carriero, N. and Gelernter, D., The S/Net's Linda kernel, in *Proceedings of the Tenth Symposium on Operating Systems Principles*, December 1985, ACM, 110.

[30] Broch, J. *et al.*, A performance comparision of multi-hop wireless ad hoc network routing protocols, in *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, Dallas, TX, USA, October 1998, ACM, 85.

[31] Tennenhouse, D.L. *et al.*, A survey of active network research, *IEEE Communications Magazine*, 35(1), 80, 1997.

© 2005 by Chapman & Hall/CRC

[32] Imielinski, T. and Goel, S., DataSpace: querying and monitoring deeply networked collections in Physical Space, *IEEE Personal Communications. Special Issue on Smart Spaces and Environments*, 7(5), 4, 2000.

[33] Bennett, F. *et al.*, Piconet: embedded mobile networking, *IEEE Personal Communications Magazine*, 4(5), 8, 1997.

[34] Heinzelman, W.R. *et al.*, Adaptive protocols for information dissemination in wireless sensor networks, in *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, Seattle, WA, USA, August 1999, ACM, 174.

[35] Adjie-Winoto, W. *et al.*, The design and implementation of an intentional naming system, in *Proceedings of the 17th Symposium on Operating Systems Principles*, Kiawah Island, SC, USA, December 1999, ACM, 186.

[36] Zhao, J. *et al.*, Residual energy scans for monitoring wireless sensor networks, in *Proceedings of the IEEE Wireless Communications and Networking Conference*, Orlando, FL, USA, March 2002, IEEE, 356, (an extended version is available as USC CSD TR-01-745), <http://www.isi.edu/scaads/papers/escan-wenc02.ps.gz>.

© 2005 by Chapman & Hall/CRC

30

Data Security Perspectives

David W. Carman

30.1 Introduction

Why do we need data security? Data generated and communicated by a sensor network can be thought of as property. Sensor network owners own this property. Like a house or car, an owner wishes to protect property from theft, damage, or destruction. Sensor network owners usually seek at least three security goals: (1) preventing data disclosure to nonowners, (2) preserving data authenticity, and (3) preserving data availability. Sensor network owners may also wish to prevent the unauthorized use of the sensor network by others (i.e. theft of service).

The consequences of not securing data in a government or military environment are severe. Losing information security battles can cause loss of military initiative, territory, resources, or even life [1]. Failure to ensure data security for sensor intelligence systems can cause grave damage to national security, sometimes for decades to come.

What makes sensor network data security unique and challenging is the unattended nature of the nodes, and the severe energy and communications constraints under which they must operate. This chapter examines common threats to sensor network data, corresponding security requirements, and the constraints of sensor environments that affect data security. We examine approaches to address these requirements and identify security mechanisms applicable to this setting.

30.2 Threats

Threats are the potential for an adversary or the environment to prevent the sensor network owners from achieving their data security goals. Passive attacks are threats that do not reveal the attacker (e.g. eavesdropping), whereas active attacks, such as transmitting electronic information or noise, alter or prevent data reception at the receiver.

The primary threat to privacy is eavesdropping using a compatible communications device. This threat can be significant for networks using common protocols such as IEEE 802.11, where millions of users have compatible wireless local-area network cards. Conversely, a system interconnected using a proprietary laser communications system will only be intercepted by very capable, well-financed, and determined adversaries.

Adversaries might use eavesdropping to thwart various sensor network goals, including avoiding detection by ''counter-detecting'' where the sensor network is located, learning when and where the

© 2005 by Chapman & Hall/CRC

sensor network has detected him, or learning the sensor network's capabilities (e.g. detection range). The adversary may obtain message contents from unencrypted communications, by thwarting any encryption of the data through cryptanalysis or other means, or by performing traffic analysis of the encrypted communications. Traffic analysis uses message size, frequency, source, and other data to garner information without knowledge of the unencrypted data traffic.

Data authenticity can be threatened by both malicious adversaries and unintentional environmental noise in the communications medium. Adversaries may attempt to forge sensor network messages by fabricating or altering the data ultimately received by the receiver. Environmental noise will cause random data errors that may prevent reconstruction of the original message at the receiver.

Adversaries may deny communications service through active jamming or subversion of network routing protocols [2]. Jamming refers to the act of generating electronic noise to prevent successful reception of communications by one or more receivers. However, jamming is inherently dangerous to the adversary, since it exposes the transmitter's whereabouts. During Operation Iraqi Freedom, the perils of an adversary attempting active jamming attacks were reinforced when U.S. Air Force Major General Victor Renuart reported that six global positioning system (GPS) jammers were located and destroyed via coalition air strike [3].

Vulnerabilities in conventional *ad hoc* routing protocols provide the opportunity for adversaries to delay or prevent sensor network traffic from reaching its intended destination [4–6]. Adversaries can forge bogus routing information to cause all data to be forwarded to an eavesdropping node. Bogus routing information can make sensor nodes believe routing paths do not exist, when in fact they do. Compromise of the *ad hoc* routing protocol may even allow an adversary to usurp the sensor network to pass its own traffic.

30.3 Security Requirements

To counter the threats to the sensor network, we describe security requirements in terms of the security services they must provide. Although data privacy and message authentication are the two main goals, we additionally examine other objectives in achieving sensor network security. A detailed treatment of general security requirements and services is provided by Stallings [7].

30.3.1 Confidentiality

Also called privacy, confidentiality is a commonly desired security service that thwarts eavesdropping by an adversary. Confidentiality is often provided by encrypting data using a cipher algorithm such as the Advanced Encryption Standard (AES) [8]. Encryption obscures the data, preventing an eavesdropper from discovering the original values without the appropriate decryption key. However, simple encryption does not prevent an eavesdropper from learning the size of messages, the frequency of the messages, and sometimes even the source and destination.

If a data-owner wishes to prevent an eavesdropper from even knowing that communications are occurring, then low probability of detection (LPD) characteristics may be desirable. LPD refers to a communications system's ability to thwart an adversary's detection of transmitted signals as information bearing. Similarly, low probability of intercept (LPI) refers to the ability to prevent an eavesdropper from determining the modulation, modulated data, or origin of detected communications. These properties are particularly desirable in military or covert intelligence environments, where the inability to detect or locate the sensor network is necessary to ensure its survival.

30.3.2 Message Authentication

Also called data origin authentication, message authentication ensures that data have not been altered from their original source. The properties of data integrity and authentication of the source together comprise the message authentication security service.

© 2005 by Chapman & Hall/CRC

Traditionally, authentication binds an assertion (e.g. ''Alice owes Bob $10'') to a person (e.g. ''Alice''). In sensor networks, authentication binds a sensor data assertion (e.g. ''Sensor #123456 sees a tank at Lat X Lon Y'') with either a general identity (e.g. ''I'm a U.S. Army, 3rd Division sensor with the network-wide key'') or a specific identity (e.g. ''I'm sensor #123456 and only I have this particular key'').

A further desirable property, that an authentic message is original and has not been replayed, is called *transaction authentication*. This property is often provided by additionally binding some time-varying parameter to the data and identity assertions (e.g. ''Sensor #123456 sees a tank at Lat X Lon Y at date/time Z''). Anti-replay protection requires the sender to apply the time-varying parameter to the message, and for the receiver to correspondingly verify the parameter using its own counter state or onboard clock.

30.3.3 Availability

The ability of sensor networks to provide their data reliably to the sensor network owners is termed availability. Sensor networks must provide the means to communicate between nodes to perform networking, and to route data through the network to the intended destination. The *ad hoc* routing protocols that enable sensor data to transit multiple hops to their destination must be protected from subversion. Since adversaries can disrupt communications at multiple layers, security mechanisms to provide availability must be allocated at each vulnerable layer.

30.3.4 Nonrepudiation

When sensor data are used for financial or legal matters, sensor data owners may need to prove the data's authenticity to third parties. *Nonrepudiation* ensures that the originator of the sensor data message cannot later deny having signed the message. Digital signatures combined with public key infrastructures provide a way of satisfying such requirements.

30.4 Constraints

Security solutions for the sensor network environment are constrained by limited battery energy, limited computational and communications capability, and the unattended nature of sensor node devices.

30.4.1 Limited Battery Energy

The computations and communications required to perform security mechanisms consume energy, reducing the lifetime of sensors with nonrechargeable batteries. Security-related computations, particularly modular exponentiations associated with public key operations, can cause the microprocessor to delay returning to an energy-conserving sleep mode. The potentially larger impact of security is the additional transmit and receive energy consumed due to communications from key management, message authentication tags, and other communicated data.

30.4.2 Limited Computational Capability

Cost, size, and energy considerations encourage sensor device designers to choose microprocessors with the minimum requirements to perform sensor processing functions. Unfortunately, these limited processors are often ill-suited to performing the computationally intensive modular exponentiation operations needed for key management and authentication. As a result, security solutions need to take into account the latency impact of cryptographic computations.

© 2005 by Chapman & Hall/CRC

30.4.3 Limited Communications Capability

Several factors limit the communications capability of sensor nodes, including:

- *Near-Earth communications paths* between ground-based sensors cause transmit signal energy to attenuate more rapidly than free-space propagation [9].
- *Small omni-directional antennas* cause antenna gain reduction [9].
- *Limited battery energy* causes budgeting of transmit signal amplification.
- *Multi-user interference*, which causes similar effects as jamming, limits the effective channel rate of each communicant.
- *Hostile jamming*, or even just the threat of it, results in waveform, receiver processing, and signal-to-jammer-ratio designs that limit throughput.

The result of these limitations is that the energy consumed per communicated bit is much greater than many conventional communication environments. When cryptographic protocols add more traffic to this communications-constrained system, the impact on energy consumption and latency can be dramatic. Latency in distributing key management information impacts the ability of the network to establish and re-establish keys in a timely fashion [10].

30.4.4 Unattended Nature of Sensor Devices

Sensor devices will almost always be located somewhere beyond the immediate control of sensor owners. This unattended nature of sensor devices makes them vulnerable to physical compromise by adversaries. Compromise of the device may include extraction of the sensor data or cryptographic keys, and may even allow the adversary to operate the device in a manner of its choosing. Compromise of cryptographic keys must be appropriately factored in to the sensor data security architecture.

30.5 Architecting a Solution

Like most disciplines, architecting a data security solution is best done by breaking a large problem into many smaller solvable problems. For a sensor network, this is typically done by allocating requisite security services to various layers of the International Standards Organization's (ISO4798) Open Systems Interconnect (OSI) model [11]. How to establish and manage the keys for all of these security services must also be planned.

Architecting an efficient security solution is highly dependent on the unique attributes of each network's sensor devices, applications, and performance requirements. Key considerations for this design are whether sensor data fusion will be required and whether sensor devices can efficiently perform public key processing.

30.5.1 Physical Layer

Allocating security services to the physical layer is generally difficult, since communications systems are routinely optimized for parameters such as throughput, error rate, bandwidth consumption, etc. Confidentiality is the sole security service routinely provided at the physical layer, and is seldom used outside the military. Sensor networks requiring LPD/LPI protection use transmission security (TRANSEC) techniques such as spreading the signal over a large bandwidth, reducing the transmit power, and/or "hopping" the frequency. When these techniques use a pseudorandom code or cryptographic key, the key management function must establish the secret value with all potential communicants prior to use.

Recently, techniques have been developed to provide authentication at the physical layer, including (1) radio frequency watermarking, where the legitimacy of the sender can be verified, and (2) message authentication streams [12], where the sender and message data can be verified as authentically bound.

© 2005 by Chapman & Hall/CRC

30.5.2 Link Layer

Sensor network security designs often allocate confidentiality and message authentication security services to the link layer. Link-layer protection protects the message from the sender to the receiver over each "hop" of a multi-hop communication. In sensor networks that fuse data at intermediate points, link-layer encryption provides protection of over-the-air traffic while allowing intermediate access for sensor data fusion. In contrast, network-layer encryption provided between two multi-hop endpoints generally prevents intermediate sensor network nodes from performing data fusion.

Message authentication at the link layer provides the important capability of verifying the legitimacy of a multi-hop message at each intermediate hop. If only network-layer message authentication is provided, then errors or forgeries will generally not be detected until the message arrives at its end destination. If verification fails at the end destination, then the resulting negative acknowledgment (NACK) and/or retransmission messages incur significant latency and consume considerable communications energy. Hop-by-hop verification helps maintain a reliable link layer that reduces end-to-end communications failures.

Both unicast and broadcast traffic can be protected at the link layer provided corresponding key management support is implemented. Unicast traffic is protected with a pairwise key shared by the sender and receiver. Broadcast traffic is protected by a group key shared by the sender and all receivers within reception range.

30.5.3 Network Layer

The network layer is a popular layer to which to allocate sensor network security services, including confidentiality, message authentication, and availability. Confidentiality and message authentication can be provided for unicast traffic by protocols such as the popular IETF standard IPsec [13]. End-to-end confidentiality at the network layer prevents intermediate nodes from accessing the sensor data. However, in cases when sensor fusion is beneficial, end-to-end encryption must be avoided. The Internet Key Exchange (IKE) [14] protocol establishes IPsec keys for pairs of communicating nodes.

Multicast traffic can be similarly protected at the network layer, although corresponding protocol standardization is less mature than for unicast traffic. No single group key management protocol is optimal for all sensor networks. For large multicast groups, hierarchical key management protocols are usually preferable, since they provide efficient additions and deletions of group members.

Availability is provided at the network layer through routing protocol security techniques. Routing protocol security is either an inherent element of the networking protocol, or a separately added protocol-specific security design. In either case, confidentiality and message authentication mechanisms are uniquely applied to each routing protocol to thwart eavesdropping and forgery by malicious network nodes. We do not examine each of these solutions here, and instead refer the reader to the efforts of the Routing Protocol Security Requirements (rpsec) IETF working group.

30.5.4 Transport Layer and Above

Most sensor networks will not have elaborate network applications operating above the network layer due to the device resource constraints. However, sensor network services above the network layer may require additional confidentiality, authentication, or access control protection.

The transport layer security (TLS) protocol standard [15], formerly called the secure sockets layer (SSL) protocol, specifies protocols and mechanisms for key management, confidentiality, and message authentication for unicast traffic. As with IKE, TLS key management incurs a significant latency and communications energy penalty that may not be suitable for lower data-rate sensor networks. The Wireless Application Protocol (WAP) Forum created a version of TLS [16] that reduces transmitted data, providing a security protocol more suitable to the wireless environment.

© 2005 by Chapman & Hall/CRC

30.5.5 Key Management

The primary security support service needed for sensor data security is *key management*. As the backbone for all security services, key management provides cryptographic keys needed for encryption, message authentication, nonrepudiation, and other security services. Key management functionality occurs, and provides keys for security services, at multiple network layers.

Key management establishes keys for pairs of communicating nodes, small groups, large groups, or even the entire sensor network. To maximize sensor data security in an environment where sensor devices are often left unattended and strong physical security is uneconomical, encryption and message authentication keys should be shared amongst the fewest nodes possible. Thus, for a unicast communication, only the sender and receiver should possess the corresponding encryption and message authentication keys. For multicast communications, the group leader should carefully manage which nodes possess the group key, including changing the key when group members join and leave. By managing keys with this fine "granularity" and changing keys regularly, the effects of an enemy compromise can be mitigated to impact only a small portion of the sensor network. Depending on the key management mechanism, however, this enhanced security may incur considerable communications and computational complexity.

Key management can be provided by a variety of mechanisms, including preloading, key transport, and key agreement. *Preloading* refers to the practice of inserting encryption and authentication keys into a device prior to it being deployed. *Key transport* involves one node securely communicating an encryption or message authentication key to one or more other nodes. *Key agreement* describes a process whereby two or more nodes engage in a protocol that establishes a common secret key. Whereas key management methods such as key transport and key agreement require some type of communications interchange to establish encryption and authentication keys between sensor nodes, traditional preloading schemes do not. Many practical key management solutions employ a mix of preloading and either key transport or key agreement.

If a key transport or key agreement scheme is used, then some type of trust management mechanism must be employed. *Trust management* comprises the set of protocols and actions necessary to ensure that parties are legitimate. Trust management steps of verifying exchanged certificates or other credentials are achieved in concert with key management as an integrated solution to establishing keys with legitimate communicants.

Physical layer TRANSEC keys are usually established via preloading, since key transport and key agreement cannot occur until communications are established. Link-layer keys may be established in a variety of ways, dependent on the requirements and constraints of the given sensor network. Network and higher layer keys are usually established via key transport or key agreement protocols such as IKE and TLS. These protocols often leverage public key infrastructures that contain certifying authorities or other trusted third parties. However, public-key-based key agreement protocols generally require several communications exchanges that incur a sizable latency and consume considerable communications energy.

30.6 Security Mechanisms

Once the required security services have been determined and their application within the communications architecture designed, the actual security mechanisms that perform the security services can be selected. Fortunately for sensor network security engineering, the maturing of the cryptography field has provided an array of algorithm choices for various security services. In this section, we discuss various security mechanisms and their suitability to sensor networks.

Despite the maturity of the cryptographic field, designing strong security mechanisms is difficult. A solid cryptography background might be sufficient to design an algorithm or protocol that has no obvious weaknesses, but almost all efforts fail to withstand the cryptanalysis of expert cryptologists. The prudent engineering approach is to use standardized algorithms and protocols for which quantitative

© 2005 by Chapman & Hall/CRC

security claims can be made. Beware the new protocol that has not been subjected to real scrutiny, or the home-grown algorithm that is patented or must remain a trade secret — such mechanisms almost always fail to provide security as strong and free as standardized mechanisms.

30.6.1 LPD/LPI

The communications capability of sensor devices will generally determine whether confidentiality can be provided by LPD/LPI techniques. Spread-spectrum modulation techniques, which are becoming common in even low-cost devices such as sensor nodes, are a practical method of providing LPD and LPI in the sensor network domain. For instance, direct-sequence spread-spectrum communications provides confidentiality when the data signal is modulated or ''spread'' by a high-rate nonlinear pseudorandom noise (PN) code known only to the sender and receiver. The PN code is commonly loaded prior to deployment, but active key management techniques could be used as well. Security for broadcast messages can be provided by sharing the PN code with all potential friendly receivers.

Frequency hopping, where the carrier frequency of the modulated signal is periodically changed, provides a deterrent to enemy interception. For this technique to work, the sender and receiver must agree on a frequency hopping pattern and maintain accurate time synchronization. The adversary must listen on a wide range of frequencies and quickly synchronize to the signal to eavesdrop effectively. Security is achieved by hopping between frequencies in a manner unknown to the adversary, over a large bandwidth, and at sufficiently fast rates that an adversary cannot gather any useful information.

In addition to modulation techniques, there are two general methods of reducing the amount of signal power received by the adversary. First, the common technique of reducing transmit power to conserve precious battery energy has the side benefit of reducing an adversary's ability to detect and intercept communications as well. Second, in the rare case where the sensor device employs a directional antenna, an adversary not in the main lobe of the transmit antenna gain will encounter a further reduction in received signal power.

30.6.2 Encryption

At the link, network, and higher layers, confidentiality is provided by transforming data using encryption algorithms. The sender employs an encryption algorithm and a key to convert the original data or *plaintext* into an unintelligible jibberish called *ciphertext*. This ciphertext is transmitted to the receiver without concern that an eavesdropper can derive any beneficial information. Upon receipt of the ciphertext, the receiver employs the corresponding decryption algorithm and key to transform the ciphertext back to plaintext. For multicast operations, the key may be established between the senders and multiple receivers.

Modern encryption algorithms come in two basic types: block ciphers and stream ciphers. Both types employ a secret key known only to the sender and receiver to transform data between plaintext and ciphertext. Secret keys that are 128 bits or more provide sufficient security for sensor data security. *Block ciphers* encrypt one block of data at a time, usually 64 or 128 bits. *Stream ciphers* create a key stream that is exclusive-ORed with 1 bit or byte of data at a time to transform data from plaintext to ciphertext.

For most sensor encryption applications, employing a block cipher is better than a stream cipher. In general, block and stream ciphers encrypt and decrypt at roughly the same speeds. From a security perspective, however, block ciphers such as Triple DES [17] and AES have been thoroughly studied for cryptographic weaknesses and are highly regarded within the research community. No stream ciphers have successfully withstood such scrutiny. This is not to say that stream ciphers such as RC4 are not secure, it is simply to point out that with all other considerations being equal, using well-analyzed block ciphers is the prudent approach. Furthermore, one of the unfortunate properties of stream ciphers is that repeated use of the same key stream allows an eavesdropper to compromise the encryption process.

© 2005 by Chapman & Hall/CRC

Although proper key management can prevent key stream reuse, there is generally no reason to choose stream ciphers over block ciphers for sensor network security.

30.6.2.1 Block Cipher Encryption Modes

Another important aspect of block cipher encryption is the choice of mode. Modes describe the method in which the block cipher algorithm and key are used to transform multiple blocks of plaintext to and from multiple blocks of ciphertext. Modes have different properties regarding error propagation, performance, and security. NIST has published a standard on the various block cipher encryption modes [45].

The most basic block cipher mode is the Electronic Code Book (ECB), where encryption is accomplished by transforming each block of plaintext into ciphertext and *vice versa*. If the amount of data to be encrypted does not equal an exact number of blocks, then a simple padding scheme is usually used to complete the remainder of the plaintext block. Upon decryption, this padding is discarded.

An undesirable security property of ECB is that two identical plaintext blocks encrypted using the same key generate the same two identical ciphertext blocks. An eavesdropper that detects the repeated ciphertext blocks at least learns that the plaintext blocks were repeated. If the eavesdropper had learned the value of one of the plaintext blocks through other means, then it could use the repetition of ciphertext blocks to learn the value of the other.

Cipher block chaining (CBC) is a block cipher encryption mode designed to overcome the repeated block vulnerability of ECB. CBC encrypts by combining the plaintext block with the previous ciphertext block in a "chain". To start the chain, an initialization vector (IV) is established by the sender and made available to the receiver — either implicitly or by explicitly transmitting it. With CBC, the chance that two identical plaintext blocks will encrypt to the same ciphertext block is exceedingly small. Although CBC is probably the most popular block encryption mode in use today, it has two notable drawbacks: (1) encryption operations cannot be performed in parallel, since each block encryption requires the result of the previous block in the chain; and (2) messages that do not end on an even block boundary require padding as described for ECB above.

Counter (CTR) mode is a relatively new block encryption mode that overcomes the two notable drawbacks of CBC. CTR uses the block cipher, a key, and a numerical counter to generate a key stream that is exclusive-ORed with the plaintext to generate the ciphertext. The receiver performs the same operation and transforms the ciphertext to plaintext. First, we note that the block encryption operations, other than the exclusive-OR, can be performed in parallel by the sender, since the input to the block cipher is known even before the plaintext. Similarly, if the receiver knows the counter and key beforehand, then it could precompute the key stream before message receipt. The second benefit of CTR is that messages that do not end on an even block boundary need not be padded. This second benefit can be especially important in resource-constrained sensor networks, where every additionally communicated bit consumes precious bandwidth and battery energy. The one major drawback of CTR mode is similar to a stream cipher: the key stream generated from a given key and counter cannot be reused. This drawback can usually be easily mitigated through proper management of the counter values.

30.6.2.2 Block Encryption Algorithms

Choosing the best block encryption algorithm has become much easier with the National Institute of Standards and Technology's (NIST) adoption of the AES in 2001. The AES block encryption algorithm provides strong confidentiality with exceptional performance on a wide range of microprocessors. AES is faster and generally regarded as more secure than Triple DES, which in turn succeeded the venerable Data Encryption Standard (DES) some years prior [17].

The security of encryption algorithms is determined by key size and the strength of the underlying algorithm to resist cryptanalytic attack. Key sizes of 128 bits and greater are sufficient to prevent brute-force searching for the correct key. Determining the quantitative security of the underlying encryption algorithm remains elusive, however. The best the current cryptographic community can

© 2005 by Chapman & Hall/CRC

currently achieve with most encryption algorithms is to demonstrate how a given algorithm is resistant to known cryptanalytic attacks such as linear [18] and differential cryptanalysis [19]. Only after years of analysis by the research community are encryption algorithms such as AES regarded as secure.

Rare exceptions to choosing AES for encryption include some military applications and some extremely memory-limited or computionally limited processors. In U.S. military applications, Type I classified algorithms might be used, but owing to their unattended nature, use of classified algorithms in sensor devices will be rare. Although not a memory ''hog,'' there may be extremely resource-constrained devices for which AES is too big or slow, and a small footprint algorithm must be used instead. Great care must be taken when considering alternate encryption algorithms, since without the proper amount of cryptanalytic investigation there can be no guarantee regarding the security of the algorithm.

30.6.3 Message Authentication

There are two main types of algorithm that provide message authentication: digital signatures and message authentication codes (MACs). Digital signatures are created by cryptographically hashing the message data and then signing the hash value using a public key algorithm. A cryptographic hash function takes a variable-sized data input and produces a small fixed-length output such as 160 bits. MD5 [20] was the preferred hash for many years, but security concerns [21] have caused most cryptographic engineers to choose the slower but more secure SHA-1 [22] for use with digital signatures. One of the main benefits of traditional digital signatures over MACs is that the signer uniquely holds the private signing key, thus enabling nonrepudiation in addition to individual source authentication of the signed message. Digital signatures are performed over sensor data or sensor network routing information just before transmission, and are appended (or prepended) to the transmitted message.

30.6.3.1 Digital Signature Algorithms

The two most popular digital signature algorithms today are the digital signature algorithm (DSA) [23] and RSA [24]. Both are public key algorithms where the basis of security is the difficulty of computing discrete logarithms (DSA) and factoring (RSA). Like AES, DSA is a NIST standard that is widely used and well studied by the cryptanalytic community. The NIST Digital Signature Standard (DSS) explicitly specifies SHA-1 hash as the accompanying hashing algorithm and generates a 320-bit signature tag regardless of the size of the corresponding public key. A major drawback for using DSA for sensor data security is that the signature generation operation performed by the sender and signature verification operation performed by the receiver are computationally intensive. Even using modern high-speed processors, millions of multiply instructions must be executed for each DSA signature generation or verification operation. For the computationally limited microcontrollers and embedded microprocessors of most sensor devices, DSA is untenable save for infrequent operations.

RSA signature generation also is computationally intensive and unattractive for routine message authentication. Unlike DSA, however, RSA signature verification is over an order of magnitude easier to compute than signature generation, thus making RSA the digital signature algorithm of choice when only verifications, such as checking certificates, will be performed by sensor devices. One additional drawback of RSA is its signature size, which is the same number of bits as the modulus, and is thus at least 1024 bits for strong security. Appending such a long signature to each message is unattractive in most sensor networks.

A variant of DSA based on computations performed over elliptic curves has been standardized by NIST and called ECDSA [23]. This public key ECDSA offers reduced computations over DSA without sacrificing security. Some methods of implementing elliptic curve operations are patented and the software to perform elliptic curve cryptography is generally more complex than DSA and RSA, but ECDSA's speed and small key size makes it an attractive alternative for digital signature operations in sensor devices. Although computationally superior to DSA and RSA, ECDSA is still a computationally expensive operation for frequent sensor data security use.

© 2005 by Chapman & Hall/CRC

30.6.3.2 MAC

MACs provide computational and bandwidth efficiency and required authentication for routine sensor data messages. Unlike public-key-based DSAs, MACs use a secret key shared by the sender and one or more receivers. There are two popular types of MAC: block-cipher-based MACs and hash-based-MACs.

Block-cipher-based MACs use a special block cipher mode to compute an authentication tag instead of ciphertext blocks. Cipher-block-chaining MACs (CBC-MACs) use the encryption algorithm, key, plaintext, and the preceding ciphertext block to compute a ciphertext block to be fed into the next link of the chain. A portion of the final ciphertext block constitutes the authentication tag. The sender generates the authentication tag and appends it to the sensor data message. The receiver computes the authentication tag using the received data, and compares the computed and received authentication tags, determining the sensor data message to be authentic if the tags match. Although block-cipher-based MACs have existed for some time, their development remains an active field of cryptographic research as increased performance and provable security claims are pursued. NIST has proposed [25] One-Key CBC-MAC (OMAC) [26] as a standard block-cipher-based MAC.

Hash-based MACs (HMACs) use cryptographic hash functions to compute the authentication tag. The sensor data to be authenticated is first hashed with an initial key to create an intermediate result. The intermediate result is then hashed with a second key to create an output that is the same size as the hash function output. The authentication tag is a truncation of the output generated by the second hash invocation.

HMACs are generally faster than block-cipher-based MACs since hash functions usually process data faster than block ciphers. Since most sensor processors are computationally limited, HMACs are the preferred choice over block-cipher-based MACs when only message authentication is being performed on the sensor data. However, if both encryption and message authentication are to be performed on a sensor data message, then at least three block-cipher-based MACs [27–29] have been designed that perform both functions in a single pass through the data. Although these *authenticated-encryption* schemes are patent pending, they are attractive for sensor security use since they require fewer computations compared with separate invocations of a block cipher for encryption and an HMAC or other block-cipher-based MAC for message authentication. The memory constraints of sensor devices may also favor the use of a single cryptographic primitive, the block cipher, over using two cryptographic primitives, a block cipher for encryption and a hash function for HMAC-based message authentication.

Transaction authentication is provided by including a monotonically increasing counter or timestamp in the MAC computation. A 32-bit or 64-bit counter maintained and sent by the sender is usually sufficient to mark each outgoing sensor data message uniquely. The receiver keeps track of the received counters, making sure the counter value always increases. Received messages that use ''old'' counter values are assumed to be replay attacks and are discarded. Accommodations for out-of-order packets that occur due to disparate network routing paths can be made using windowing methods suggested by Kent and Atkinson [13]. The sender may also include a timestamp in each sensor data message that is verified by the receiver. Some level of clock synchronization is needed in this case to avoid discarding legitimate messages due to the sender or receiver having an inaccurate clock.

30.6.4 Key Management

A critical component of encryption and message authentication security is how the mechanism's key is established and maintained. A small key that is fixed for a long period of time and shared with a large number of sensor nodes provides less security than a large key that is changed often and shared among only the few sensor nodes that need it. Larger keys are more secure than smaller keys (to a point), since larger keys require an attacker to attempt more keys to guess the key by brute force. Most military systems use cryptoperiods, where a given cryptographic key is used for only a limited period of time.

© 2005 by Chapman & Hall/CRC

Active key management, where the key is established by the sensor nodes themselves while deployed, is an effective way of limiting access to the key.

30.6.4.1 Preloading

The simplest method of providing needed keys to sensor devices is *preloading* the entire sensor network with the same encryption and MAC keys. Preloading refers to the practice of loading keys into the sensor nodes prior to them being deployed. A major advantage of preloading is that little or no key management messages need be exchanged during deployment. However, there are several disadvantages of preloading, including the susceptibility to compromise when using a networkwide key and the inability to change keys in the advent of such a compromise. Two preloading schemes [30,31] avoid use of a networkwide key by randomly predistributing many keys to subsets of nodes in a manner that probabilistically assures that any two communicants will share a common key, while dramatically reducing the potential exposure of each key. A disadvantage of the random preloading schemes is that discovery of which keys the other communicant possesses requires energy and time-consuming communications to occur.

30.6.4.2 Public key cryptography

To achieve a greater resistance to compromise, security architectures can forgo the use of a widely deployed key and instead use key management protocols based on public key cryptography. A public key cryptography system uses two keys: a widely known public key and a private key known only by the possessor. The public and private keys are mathematically related, but in such a way that prevents an adversary from determining the private key from the public key. Most public key cryptography systems are based on one of three hard mathematical problems: (1) integer factorization, which is the security basis of RSA; (2) discrete logarithm, which is the security basis of Diffie–Hellman key agreement; or (3) elliptic curve cryptography, which is the security basis of the elliptic-curve-based Diffie–Hellman key agreement.

RSA and Diffie–Hellman are the most popular public key methods and are the basis for most modern key management protocols such as IKE [14] and TLS [15]. These protocols interactively exchange public keys and other key management information and can be used to establish a key *after* sensor devices are deployed. Established keys are unique to the communicants that hold the private keys and participate in the protocol, and are thus less susceptible to disclosure. Key management protocols based on public key algorithms may be invoked more than once, changing encryption and authentication keys regularly to limit the amount of data protected, and thus limiting the amount of sensor data that could be compromised.

For public keys to be useful, nodes must trust the association between a public key and an entity. Modern key management protocols leverage the use of public key infrastructures that allow nodes to trust the binding between the identity of a node and its public key. This cryptographic binding of information, such as identity, role, or other attributes, is instantiated in a public key *certificate*. A unique certificate is generated for each sensor device by the sensor data owner, who acts as a *certifying authority* for the sensor network. Sensor devices usually obtain the public key certificate via an online exchange with the node to which the certificate refers. Eschenauer *et al.* [32] suggest alternate methods of generating, distributing, and discovering trust information. Stajano and Anderson [33] offer a radical approach that may be practical for sensor networks where nodes spend the majority of their time in an energy-conserving sleep state.

Once trust is established, key agreement or key transport protocols may be used to establish keys. Key agreement allows both nodes to contribute key management information that is used to create the key. Mutual contributions prevent one node from causing a weak key to be used, whether intentionally or unintentionally. Key transport occurs when one node generates a secret key and sends it to another node. RSA is most often used to protect key transport by encrypting the secret key using the public key of the recipient. A version of Diffie–Hellman called ElGamal [34] may also be used for this purpose.

© 2005 by Chapman & Hall/CRC

Although key transport is not contributory as with key agreement, it can reduce the number of interactive key management messages that must be exchanged.

RSA and Diffie–Hellman require significant computations, making them unattractive for most sensor network applications. Both protocols perform modular exponentiations that require millions of multiply instructions on the computationally limited microprocessors present in most sensor devices. As a result, RSA and Diffie–Hellman operations sometimes take seconds (or longer) to be performed, thus significantly contributing to the latency in establishing keys. Since most sensors return their processors to an energy-conserving sleep state when not being used, the additional computation time also consumes more energy.

A potentially larger issue than computations is the energy consumed and latency incurred due to the communications required by the RSA and Diffie–Hellman protocols. The public key certificates and other public key management information communicated require several thousands of bits to be exchanged. These communications consume both transmit energy by the sender and receive energy by the receiver. Moreover, a considerable latency is introduced due to the time required to send the information between nodes. This latency is actually much greater than the number of bits multiplied by the data rate, since the effects of the media access control layer and retransmissions must also be considered. The sum of computational and communications effects on energy consumed and latency incurred usually render these protocols unsuitable for use in sensor networks.

Elliptic curve cryptography (ECC) [35] is an attractive alternative to RSA and Diffie–Hellman for sensor key management. Diffie–Hellman key agreement based on elliptic curve operations requires fewer computations and communications for the same level of security. For instance, a 161-bit ECC key is roughly equivalent in security strength to a 1024-bit RSA key [36]. Minimal certificates for ECC systems are about 62 bytes, whereas RSA systems are 256 bytes [36]. ECC-based schemes have been added to IKE and other standard protocols as these techniques have matured and gained wider acceptance.

Identity-based cryptography [37] is especially attractive for use in sensor networks since it significantly reduces key management messaging. A trusted authority generates a private key for each sensor node based on its identity. Only the trusted authority knows how to perform this function. Sensor nodes derive public keys from sensor device identities, thus eliminating the exchange of public key certificates when executing a key management protocol. Maurer and Yacobi [38] described a scheme [8] that eliminates the need to exchange anything other than the identities. Although their scheme requires significant offline computations for private key generation, Matt [39] provides an overview of recently developed identity-based elliptic curve schemes that do not suffer from this problem.

30.6.4.3 Group key management

When encryption or message authentication is required to secure broadcast or multicast communications, a group key management protocol is needed. Few group key management protocols have been standardized, let alone targeted for sensor network environments. Group key agreement protocols, such as Group Diffie–Hellman [40] and Burmester–Desmedt [41] require considerable communications, even when ECC keys are used. A group keying scheme that combines identity-based cryptography and group key transport was invented by Matt and examined for use in an Army sensor network [42].

30.7 Other Sources

An excellent scientific treatment of the theory and application of cryptography can be found in Handbook of Applied Cryptography [43]. Applied Cryptography [44] provides a very good introduction to cryptography with an encyclopedia of references to cryptographic algorithms, protocols, and applications.

© 2005 by Chapman & Hall/CRC

30.8 Summary

How best to provide data security for a given sensor network is highly dependent on the unique requirements and constraints of the target environment. However, we have identified approaches and standard mechanisms that provide effective data security against a variety of threats to data security. Encryption and message authentication are best provided at the link and network layers, depending on whether sensor data fusion is being performed. When sensor data fusion is needed, hop-by-hop encryption and message authentication should be implemented. When data fusion is not required, hop-by-hop authentication and end-to-end encryption and message authentication are an effective multi-layer protection.

When selecting security algorithms, a compelling reason is needed *not* to use AES for encryption. HMAC-SHA-1, HMAC-MD5, and AES-based OMAC provide strong authentication of routine sensor data messages when keys of at least 128 bits are used. ECDSA should be used to sign data during infrequent but important operations, such as key management, routing, and special sensor data operations.

Key management and routing protocol security are challenging due to the limited bandwidth, energy, and unattended nature of nodes in the sensor network environment. Strong key management requires occasional exchange of key management messages and public key computations, which consume energy and incurs latency in resource-constrained sensor devices. ECC, whether used in conjunction with a public key infrastructure or using an identity-based scheme, provides an efficient alternative to the popular RSA and Diffie–Hellman algorithms. Group key management, needed to secure link-layer broadcast or network-layer multicast, is even more challenging. Conventional *ad hoc* routing protocols are subject to subversion and need protocol-specific protections to provide sensor data availability successfully.

References

[1] Denning, D., *Information Warfare and Security*, Addison-Wesley, 1999.

[2] Wood, A. and Stankovic, J., Denial of service in sensor networks, *IEEE Computer*, 35(10), 54, 2002.

[3] Wilkison, R., *U.S. Says Iraq GPS Jamming Sites Destroyed*, Iraq Crisis Bulletin/(INEWS) International News E-Wire Service, Doha, Qatar, March 26, 2003.

[4] Law, Y. *et al.*, Assessing security-critical energy-efficient sensor networks, TR-CTIT-02-18, June 2002.

[5] Papadimitratos, P. and Haas, Z., Secure routing for mobile *ad hoc* networks, in *SCS Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS 2002)*, San Antonio, TX, January 27–31, 2002.

[6] Murphy, S. and Weiler, S., Overview of potential compromises and security paradigms in wireless routing protocols, in *Proceedings of Collaborative Technology Alliance Conference 2003*, College Park, MD, 2003.

[7] Stallings, W., *Cryptography and Network Security: Principles and Practice*, 3rd ed., Prentice Hall, 2003.

[8] U.S. Department of Commerce, National Institute of Standards and Technology (NIST), *Advanced Encryption Standard (AES) (Federal Information Processing Standards Publication 197)*, November 2001.

[9] Asada, G. *et al.*, Wireless integrated network sensors: low power systems on a chip, in *European Solid State Circuits Conference*, The Hague, Netherlands, October 1998.

[10] Carman, D. and Cirincione, G., Energy and latency costs of communicating certificates during secure network initialization of sensor networks, in *Proceedings of Collaborative Technology Alliance Conference 2003*, College Park, MD, 2003.

[11] International Organization for Standardization (ISO), *Information Processing Systems Open Systems Interconnection Basic Reference Model*, ISO 7498, 1984.

© 2005 by Chapman & Hall/CRC

[12] Carman, D. and Boncelet, C., A new message authentication approach with less overhead and greater reliability, in *Proceedings of Collaborative Technology Alliances (CTA) Communications & Networks (C&N) Alliance 2003 Annual Symposium*, April 2003.
[13] Kent, S. and Atkinson, R., Security architecture for the Internet protocol, RFC 2401, November 1998.
[14] Harkins, D. and Carrel, D., The Internet key exchange (IKE), RFC 2409, November 1998.
[15] Dierks, T. and Allen, C., The TLS protocol version 1.0, RFC 2246, January 1999.
[16] Wireless Application Protocol Forum, Ltd., WAP transport layer end-to-end security, 2001.
[17] U.S. Department of Commerce, National Institute of Standards and Technology (NIST), *Data Encryption Standard (Federal Information Processing Standards Publication 46–3)*, October 1999.
[18] Matsui, M., Linear cryptoanalysis method for DES cipher, in *Proceedings of EUROCRYPT 1993*, Springer-Verlag, 1993, 386.
[19] Biham, E. and Shamir, A., *Differential Cryptanalysis of the Data Encryption Standard*, Springer-Verlag, 1993.
[20] Rivest, R., The MD5 message-digest algorithm, RFC 1321, April 1992.
[21] Dobbertin, H., Cryptanalysis of MD5 compress, German Information Security Agency, May 1996.
[22] U.S. Department of Commerce, National Institute of Standards and Technology (NIST), *Secure Hash Standard (Federal Information Processing Standards Publication 180–2)*, August 2002.
[23] U.S. Department of Commerce, National Institute of Standards and Technology (NIST), *Digital Signature Standard (DSS) (Federal Information Processing Standards Publication 186-2)*, January 2000.
[24] RSA Laboratories, PKCS #1 v2.1: RSA cryptography standard, June 2002.
[25] National Institute of Standards and Technology (NIST), Rationale for the selection of the OMAC variation of XCBC, 2003.
[26] Iwata, T. and Kurosawa, K., OMAC: One-Key CBC MAC, in *Proceedings of Fast Software Encryption 2003*, *Lecture Notes in Computer Science*, Springer-Verlag, 2003.
[27] Jutla, C., Encryption modes with almost free message integrity, *Advances in Cryptology — EUROCRYPT '01*, *Lecture Notes in Computer Science*, Vol. 2045, Springer-Verlag, 2001.
[28] Gligor, V. and Donescu, P., Fast encryption and authentication: XCBC encryption and XECB authentication modes, presented at the *2nd NIST Workshop on AES Modes of Operation*, Santa Barbara, CA, August 24, 2001.
[29] Rogaway, P. *et al.*, OCB: a block-cipher mode of operation for efficient authenticated encryption, in E*ighth ACM Conference on Computer and Communications Security (CCS-8)*, ACM Press, 2001, 196.
[30] Eschenauer, L. and Gligor, V., A key-management scheme for distributed sensor networks, in *Proceedings of the 9th ACM Conference on Computer and Communications Security 2002*, Washington, DC, USA.
[31] Perrig, A. *et al.*, Random key predistribution schemes for sensor networks, in *IEEE Symposium on Security and Privacy*, 2003.
[32] Eschenauer, L. *et al.*, On trust establishment in mobile *ad-hoc* networks, in *Proceedings of the Security Protocols Workshop*, Cambridge, UK, April 2002.
[33] Stajano, F. and Anderson, R., The resurrecting duckling: security issues for *ad-hoc* wireless networks, in *Security Protocols*, *7th International Workshop*, 1999.
[34] ElGamal, T., A public key cryptosystem and a signature scheme based on the discrete logarithm, *IEEE Transactions on Information Theory*, 31(4), 469, 1985.
[35] Menezes, A., *Elliptic Curve Public Key Cryptosystems*, Kluwer Academic Publishers, 1993.
[36] Johnson, D., *ECC*, *Future Resiliency and High Security Systems*, Certicom Corporation, March 1999.
[37] Shamir, A., Identity-based cryptosystems and signature schemes, in *Proceedings of Crypto'84*, 1985, 47.

© 2005 by Chapman & Hall/CRC

[38] Maurer, U. and Yacobi, Y., A non-interactive public-key distribution system, *Designs*, *Codes and Cryptography*, 9(3), 305, 1996.
[39] Matt, B., Efficient pairwise key establishment for battlefield networks, in *Proceedings of Collaborative Technology Alliance Conference 2003*, College Park, MD, 2003.
[40] Steiner, M. *et al*., Key agreement in dynamic peer groups, *IEEE Transactions on Parallel and Distributed Systems*, 11(8), 769, 2000.
[41] Burmester, M. and Desmedt, Y., A secure and efficient conference key distribution system, in *Proceedings of Eurocrypt '94*, *Lecture Notes in Computer Science 950*, Springer-Verlag, 1995, 275.
[42] Carman, D. *et al*., Energy-efficient and low-latency key management for sensor networks, in *Proceedings of the 23rd Army Science Conference*, December 2002.
[43] Menezes, A. *et al*., *Handbook of Applied Cryptography*, CRC Press, 1996.
[44] Schneier, B., *Applied Cryptography*, 2nd ed., John Wiley & Sons, 1996.
[45] Dworkin, M, *NIST Special Publication 800-38A — Recommendation for Block Cipher Modes of Operation — Methods and Techniques*, National Institute of Standards and Technology (NIST), December 2001.

© 2005 by Chapman & Hall/CRC

31

Quality of Service Metrics

N. Gautam*

31.1 Service Systems

The phrase ''quality of service'' (QoS) has been popular for about 15 years; however, there has been little or no consensus in terms of what QoS actually is, what various QoS metrics are, and what QoS specifications are. Yet, QoS has spread far and wide, beyond the realm of networking, into areas such as transportation, health care, hospitality, manufacturing, etc. In fact, this author believes it may be better to introduce QoS using examples from the service industry to provide appropriate analogies in the hope of giving the study of QoS more structure, as well as to discover newer ways of providing QoS in computer networks. To define a service industry, one must first differentiate between goods, which are usually tangible, and services, which typically are intangible. In fact, several organizations that have been traditionally concentrating on their goods (such as cars at vehicle manufacturers, food at restaurants, books at bookstores, etc.) are now paying a lot of attention to service (such as on-time delivery, availability, warranties, return policies, overall experience, etc.). These typically fall under the realm of providing QoS.

31.1.1 Elements of a Service System

Examples of service systems range from complex interconnected networks such as computer-communication networks, transportation systems, theme parks, etc. to simpler individual units such as a barber shop, repair shops, theaters, restaurants, hospitals, hotels, etc. (Figure 31.1). In all these examples two key players emerge, namely the service provider and users. As the names suggest, the users receive service provided by the service provider. Users (also called customers if there is money involved) do not necessarily have to be humans; they could be other living or nonliving entities. Further, users do not have to be single individuals, they could be part of a group (such as in a multicast session, in a restaurant, at a play, etc.). By the same token, for a given system, there could be zero, one, or many

*The author was partially supported by NSF grant ANI-0219747.

© 2005 by Chapman & Hall/CRC



Figure 31.1. Service systems.

service providers. Although most services are such that they are owned by a single entity (the one to blame if things go wrong), there are some (including the Internet) that are owned by several groups.

QoS can be defined as a set of measures that the users ''want'' from the system (or sometimes what the service provider wants to give the users). What the users eventually ''get'' is called performance. From a physical goods standpoint, QoS is equivalent to specifications (or specs as they are usually abbreviated). Some QoS measures are qualitative (such as taste, ambience, etc.) and these are hard to provide, since different users perceive them differently. Other QoS measures which are quantitative also have some fuzziness attached. For example, on one day a user might find a 90 ms latency intolerable and on another day the user may find 100 ms latency tolerable. There could be several reasons for that, including the mood of the user, the expectations of the user, etc. Capturing such cognitive aspects are beyond the scope of this chapter. We will focus on systems where user requirements (i.e. QoS) are known precisely and users are satisfied or unsatisfied if the requirements are met or not met respectively. That means if 100 ms is the tolerance for latency, then QoS is met (not met) if latency is lesser (greater) than 100 ms.

In some service systems the users and the service providers negotiate to come up with what is known as a service-level agreement (SLA). For example, years ago a pizza company promised to deliver pizzas within 45 min, or the pizzas are free. That is an example of an SLA which is also called a QoS guarantee. In many service systems there is no explicit guarantee, but a QoS indication such as: ''your call will be answered in about three minutes,'' ''the chances of a successful surgery is 99.9%,'' ''the number of defective parts is one in a million,'' etc. In many systems it is not possible to provide absolute QoS guarantees due to the dynamic nature of the system, but it may be feasible to deliver a relative QoS. This is typically known as the level of service (LoS); for example, if there are three LoSs (gold, silver and

© 2005 by Chapman & Hall/CRC

bronze), then, at a given time instant, gold will get a better QoS than silver, which would get a better QoS than bronze.

31.1.2 Customer Satisfaction

Although many consider QoS and customer satisfaction as one and the same, QoS here is thought of as only a part of customer satisfaction. However, it is not assumed here that providing QoS is the objective of a service provider, but providing customer satisfaction is. With that understanding, the three components of customer satisfaction are (a) QoS, (b) availability, and (c) cost. The service system (with its limited resources) can be considered either physically or logically as one where customers arrive, if resources are available they enter the system, obtain service for which the customers incur a cost, and then they leave the system (see Figure 31.2). One definition of availability is the fraction of time arriving customers enter the system. Thereby, QoS is provided only for customers that ''entered'' the system. From an individual customer's standpoint, the customer (i.e. user or application) is satisfied if the customer's requirements over time on QoS, availability and cost are satisfied. Some service providers' objective is to provide satisfaction aggregated over all customers (as opposed to providing absolute service to an individual customer). Services such as restaurants provide both: on the one hand, they cater to their frequent customers and on the other hand they provide overall satisfaction to all their customers.

The issue of QoS (sometimes called conditional QoS, as the QoS is conditioned upon the ability to enter the system) versus availability needs further discussion. Consider the analogy of visiting a medical doctor. The ability to get an appointment translates to availability; however, once an appointment is obtained, QoS pertains to the service rendered at the clinic, such as waiting time, healing quality, etc. Another analogy is airline travel. Getting a ticket on an airline at a desired time from a desired source to a desired destination is availability. QoS measures include delay, smoothness of flight, in-flight service, etc. One of the most critical business decisions is to find the right balance between availability and QoS. The two are inversely related, as illustrated in Figure 31.3. A service provider can increase availability by

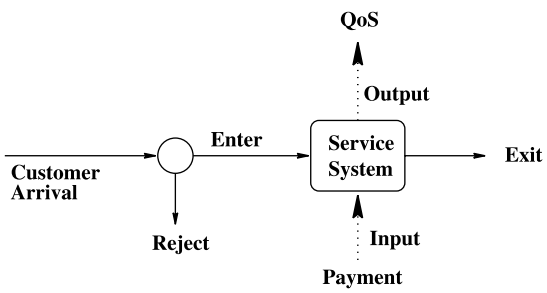


Figure 31.2. Customer satisfaction in a service system.

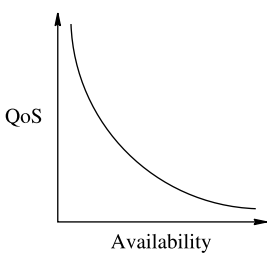


Figure 31.3. Relationship between QoS and availability, given cost.

© 2005 by Chapman & Hall/CRC

decreasing QoS and *vice versa*. A major factor that could affect QoS and availability is cost. Usually with cost (somewhat related to pricing) there is a need to segregate the customers into multiple classes. It is not necessary that classes are based on cost, they could also depend on customer type (i.e. applications) and QoS requirements. The ability to provide appropriate customer satisfaction based on class (and relative to other classes) is a challenging problem especially under conditions of stress, congestion, unexpected events, etc. For example, if an airplane encounters turbulence, all customers experience the bumpy ride, irrespective of the class of service.

31.1.3 Effect of Resources and Demand

Customer satisfaction is closely related to both resources available at the service provider and demand from the customers. It is of grave importance to understand the relationship and predict or estimate it. Firstly, consider the relationship between resource and performance. The graph of resources available at a service provider versus the performance the service provider can offer is usually as described in Figure 31.4. From Figure 31.4 the following are evident: (1) it is practically impossible to get extremely high performance, and (2) to get a small increase in performance it would sometimes even require twice the amount of resources, especially when the available performance is fairly high in the first place.

Now consider the relationship between customer satisfaction and demand. If a service offers excellent customer satisfaction, very soon its demand would increase. However, if the demand increases, then the service provider would no longer be able to provide the high customer satisfaction, which eventually deteriorates. Therefore, demand decreases. This cycle continues until one of two things happens: either the system reaches an equilibrium or the service provider goes bankrupt. The situations are depicted in Figure 31.5. Notice the relation between customer satisfaction and demand: they are inversely related from the service provider's standpoint and directly related from a customer's standpoint.

After studying some general aspects of providing QoS in the service industry, we now turn our attention to QoS provisioning in networking.

31.2 QoS in Networking

Although the Internet started as a free and best-effort service, the next-generation Internet is showing clear signs of becoming a network based on both pricing issues and QoS. The concept, need, and application of QoS has also quickly spread to other high-speed networks, such as telephony, peer-to-peer networks, sensor networks, *ad hoc* networks, private networks, etc. For the remainder of this

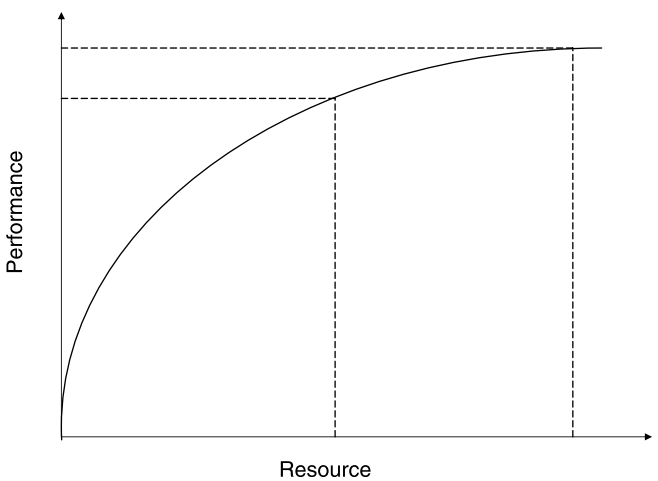


Figure 31.4. Resource versus performance.

© 2005 by Chapman & Hall/CRC

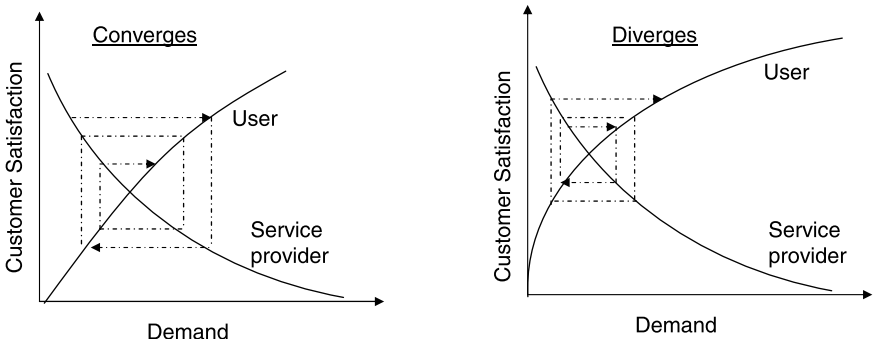


Figure 31.5. Demand versus customer satisfaction.

chapter on QoS, we will use networking rather loosely in the context of any of the above networks. We will not be restricting ourselves to any particular network or protocols running on the network. It may be best to think of an abstract network with nodes (or vertices) and arcs (or links). In fact the nodes and arcs can be either physical or logical in reality. At each node there are one or more queues and processors that forward information. With this setting in mind, we proceed to investigate the broad topic of QoS in high-speed networks.

31.2.1 Introduction

The plain old wired telephone network, one of the oldest computer-communication networks, is one of the only high-speed networks that can provide practically perfect QoS, at least in the United States. One of the positive aspects of this is that give other networks (such as the Internet) time and they would eventually become as effective as the telephone networks in terms of providing QoS. Having said that, it is important to notice that in some cases the telephone network does struggle to provide QoS. One example is during the terrorist attacks on September 11, 2001: it was impossible to dial into several areas. Another example is international phone calls: there is a long way to perfecting QoS for that. With the advent of cell phones, the telephone networks are now facing a new challenge of providing QoS to wireless customers.

From a networking standpoint (for Internet-type networks), one of the difficulties for providing QoS is the presense of multiple classes of traffic, such as voice, video, multimedia, data, web, etc. Unlike wired telephone networks that offer a bandwidth of about 60 kbps for whatever type of call (regular phone calls, fax, modem call, etc.), in networking the various types of traffic have varying requirements. In fact, real-time traffic can tolerate some loss but very little delay; however, non-real-time traffic cannot tolerate loss but can take a reasonable amount of delay. The number of network applications is increasing steadily; however, it is not practical to support more than a handful of classes (two to four classes). Therefore, clever traffic aggregation schemes need to be developed.

In order to provide QoS in a multi-class network, some of the important aspects to consider and optimize are: scheduling, switching, and routing. For differentiating the various classes of traffic, as well as providing different QoS, information needs to be processed in a manner other than first-come-first-served (FCFS). The essence of scheduling is to determine what information to serve next. The telephone network, which essentially has only one class of traffic, does not do any special scheduling. It just does switching and routing. Switching is done using circuit-switching policies where, upon dialing a number, a virtual path is created from the source to the destinations through which information is transmitted. An appropriate routing algorithm is used to send information from the source to the destination efficiently. From a networking standpoint, in doing appropriate scheduling, switching, and routing it would be possible to provide QoS to the users. How to do that is being pursued actively by the research community.

© 2005 by Chapman & Hall/CRC

31.2.2 Characteristics of Network QoS Metrics

Before looking at how to provision QoS in networking applications, it is important to understand what the QoS metrics are in the first place and how to characterize them. There are four main QoS metrics in networking: delay, jitter, loss and bandwidth.

- *Delay*. This is defined as the time elapsed between when a node leaves a source and reaches a destination. Though the term delay implies there is a target time and the information comes after the target time elapses, that really is not the case. It is just a measure of travel time from source to destination, which is also called latency or response time.
- *Jitter*. The variation in the delay is termed as jitter. If a stream of packets is sent from a source to a destination, then typically all packets do not face the same delay. Some packets experience long delays and others experience short delays. Applications such as video-on-demand can tolerate delays but not jitter. A simple way of eliminating or reducing jitter is to employ a jitter buffer at the destination. All packets are collected and then transmitted. This does increase the delay, however.
- *Loss*. When a piece of information arrives at a node at a time when the queue at the node is full (i.e. full buffer), then the information is dropped (or lost). This is known as loss. There are several measures of loss, including loss probability (the probability that a piece of information can be lost along its way from its source to its destination) and loss rate (the average amount of information lost per unit time in a network or node).
- *Bandwidth*. Several real-time applications, such as voice over IP, video-on-demand, etc., require a certain bandwidth (in terms of bytes per second) to be available for successful transmission. In fact, the only QoS guarantee a telephone network provides is bandwidth (of about 60 kbps).

Note: as alluded to in Section 31.1.2, while studying QoS, the concept of availability is skipped; however, it is very important from a customer-satisfaction standpoint to consider availability.

Performance metrics can be typically classified into three parts: additive, multiplicative and minimal. In order to explain them, consider a traffic stream that originates at a particular node, and traverses through N nodes before reaching its destination. The objective is to obtain end-to-end performance metrics given metrics across nodes. For example consider node i (for $i \in [1, N]$), let d_i, ℓ_i and b_i respectively be the delay, loss probability, and bandwidth across node i. Assume that the performance metrics across node i are independent of all other nodes. To compute end-to-end performance measures the following are used.

- *Additive*. The end-to-end performance measure is the sum of the performance measures over the individual nodes along the path or route. The end-to-end delay D for our above example is obtained as

$$D = d_1 + d_2 + \cdots + d_N = \sum_{i=1}^{N} d_i$$

- *Multiplicative*. The end-to-end performance measure is the product of the performance measures over the individual nodes along the path or route. The end to end loss L for our above example is obtained as

$$L = 1 - (1 - \ell_1)(1 - \ell_2)\ldots(1 - \ell_N)$$

  Note that the multiplicative metric can be treated as an additive metric by taking the logarithm of the performance measure.
- *Minimal*. The end-to-end performance measure is the minimum of the performance measures over the individual nodes along the path or route. The end-to-end bandwidth B for our above

© 2005 by Chapman & Hall/CRC

example is obtained as the minimum bandwidth available across all the nodes in its path. In particular:

$$B = \min\{b_1, b_2, \ldots, b_N\}$$

Although all performance metrics are inherently stochastic and time varying, in order to keep the analysis tractable the following are typically done: replace a metric by its long-run, steady state, or stationary equivalent; use an appropriate deterministic value, such as maximum, minimum, mean, median, or mode; use a range of meaningful values. Now, in order to guarantee QoS, depending on whether a deterministic or stochastic performance metric is used, the guarantees are going to be either absolute or probabilistic respectively. For example, you could give an absolute guarantee that the mean delay is going to be less than 100 ms. Or you could say that the probability that the delay is greater than 200 ms is less than 5%. It is also possible to get bounds on the performance, and it is important to note that giving deterministic bounds could mean underutilization of resources and, thereby, very poor availability. Once again, it is crucial to realize that, for a given infrastructure, the better QoS guarantees one can provide, the worse off will be availability (see Figure 31.3).

31.3 Systems Approach to QoS Provisioning

In this section we focus on obtaining the performance metrics that form the backbone of QoS analysis. There are several methodologies to evaluate the performance of a system, and they can be broadly classified into experimental, simulation-based, and analytical methods. Experimental methods tend to be expensive and time consuming, but they require very little approximations. The analytical models are just the opposite. Simulation-based techniques fall in the middle of the spectrum. This chapter focuses on obtaining analytical results which would be mainly used in making optimal design and admission control decisions. These can be appropriately used for strategic, tactical, and operational decisions, depending on the time granularity. In this section we present two main performance analysis tools based on queueing theory and large deviations theory.

31.3.1 Performance Analysis Using Queueing Models

We begin this section by considering a single station queue and then extend the theory to a network of queues. From an analysis standpoint, the most fundamental queueing system is the $M/M/1$ queue. Input to the queue is according to a Poisson process with average rate λ per unit time (i.e. inter-arrival times exponentially distributed). Service times are exponentially distributed with mean $1/\mu$. It is a single server queue with infinite waiting room and FCFS service. The following performance measures can be derived (under the assumption $\lambda < \mu$): average number in the system is $\lambda/(\mu - \lambda)$ and average waiting time in the system is $1/(\mu - \lambda)$. For the $M/M/1$ queue, the distribution of the waiting times is given by

$$P\{\text{Waiting Time } \leq x\} = 1 - \mathrm{e}^{-(\mu-\lambda)x}$$

In this way, other generalizations to this model, such as a different arrival process, a different service time distribution, more servers, finite waiting room, different order of service, etc., can be studied. The reader is referred to one of several standard texts on queues (such as Gross and Harris [1] and Bolch *et al.* [2]).

Now we turn to a network of queues, specifically what is known as a *Jackson network*. The network consists of N service stations (or nodes). There are s_i servers at node i. Service times at node i are exponentially distributed and independent of those at other nodes. Each node has infinite waiting room. Externally, customers arrive at node i according to a Poisson process with mean rate θ_i. Upon completion of service at node i a customer departs the system with probability r_i or joins the queue at

© 2005 by Chapman & Hall/CRC

node j with probability p_{ij}. Assume that at least one node has arrivals externally and at least one node has customers departing the system. The vector of effective arrival rates $\lambda = (\lambda_1 \ \lambda_2 \ \ldots \ \lambda_N)$ can be obtained using

$$\lambda = \theta(I - P)^{-1}$$

where $\theta = (\theta_1 \ \theta_2 \ \ldots \ \theta_N)$, P is the routing probability matrix composed of various $[p_{ij}]$ values and I is an $N \times N$ identity matrix. Then, each queue i can be modeled as independent single station queues. Jackson's theorem states that the steady-state probability of the network can be expressed as the product of the state probabilities of the individual nodes. An application of Jackson networks is illustrated in Section 31.4.1.

31.3.2 Performance Analysis Using Large Deviations Theory

In this section we will focus on using the principle of large deviations for performance analysis of networks based on fluid models. Although large deviations does not require fluid traffic, the reason we pay attention to it is that fluid models represent correlated and long-range dependent traffic very well. In fact, the simple on–off source could be thought of as one that generates a set of packets back to back when it is on and nothing flows when it is off.

Let $A(t)$ be the total amount of traffic (fluid or discrete) generated by a source (or flowing through a pipe) over time $(0, t]$. For the following analysis, consider a fluid model. Note that it is straightforward to perform a similar analysis for discrete models as well. However, the results will be identical. Consider a stochastic process $\{Z(t), t \geq 0\}$ that models the traffic flow. Also, let $r(Z(t))$ be the rate at which the traffic flows at time t. Then

$$A(t) = \int_0^t r(Z(u))\,\mathrm{d}u$$

The asymptotic log moment generating function (ALMGF) of the traffic is defined as

$$h(v) = \lim_{t\to\infty} \frac{1}{t} \log E\{\exp(vA(t))\}$$

Using the above equation, it is possible to show that $h(v)$ is an increasing, convex function of v and for all $v > 0$

$$r^{\text{mean}} \leq h'(v) \leq r^{\text{peak}}$$

where $r^{\text{mean}} = E(r(Z(\infty)))$ is the mean traffic flow rate, $r^{\text{peak}} = \sup_z\{r(z)\}$ is the peak traffic flow rate, and $h'(v)$ denotes the derivative of $h(v)$ with respect to v. The *effective bandwidth* of the traffic is defined as

$$\text{eb}(v) = \lim_{t\to\infty} \frac{1}{vt} \log E\{\exp(vA(t))\} = h(v)/v$$

It can be shown that $\text{eb}(v)$ is an increasing function of v and

$$r^{\text{mean}} \leq \text{eb}(v) \leq r^{\text{peak}}$$

© 2005 by Chapman & Hall/CRC

Also

$$\lim_{v\to 0} \text{eb}(v) = r^{\text{mean}} \quad \text{and} \quad \lim_{v\to\infty} \text{eb}(v) = r^{\text{peak}}$$

It is not easy to calculate effective bandwidths using the formula provided above. However, when $\{Z(t),\ t \geq 0\}$ is a continuous-time Markov chain (CTMC) [3,4], or a semi-Markov process [5], one can compute the effective bandwidths more easily. Also, see Krishnan *et al.* [6] for the calculation of effective bandwidths for traffic modeled by a fractional Brownian motion.

Consider a single buffer fluid model as depicted in Figure 31.6. Input to a buffer of size B is driven by a random environment process $\{Z(t),\ t \geq 0\}$. When the environment is in state $Z(t)$, fluid enters the buffer at rate $r(Z(t))$. The output capacity is c. Let $X(t)$ be the amount of fluid in the buffer at time t. We are interested in the limiting distribution of $X(t)$, i.e.

$$\lim_{t\to\infty} P\{X(t) > x\} = P\{X > x\}$$

Assume that the buffer size is infinite. In reality, the buffer overflows (hence packets/cells are lost) whenever $X(t) = B$ and $r(Z(t)) > c$. Note that the buffer content process $\{X(t),\ t \geq 0\}$ (when $B = \infty$) is stable if the mean traffic arrival rate is less than c, i.e.

$$E\{r(Z(\infty))\} < c$$

Then $X(t)$ has a limiting distribution. From the limiting distribution, use $P\{X > B\}$ as an upper bound for loss probability (remember that B is the actual buffer size). This can also be used for delay of QoS. Fluid arriving at time t waits in the buffer (hence faces a delay) for $X(t)/c$ amount of time. Therefore, the long-run probability that the delay across the buffer is greater than δ is

$$P\{\text{Delay} > \delta\} = P\{X > c\delta\}$$

Using results from large deviations, it is possible to show that

$$P\{X > x\} \approx \text{e}^{-\eta x}$$

for large values of x (specifically as $x \to \infty$), where η is the solution to

$$\text{eb}(\eta) = c$$

Note that the above expression is an approximation, and in fact researchers have developed better approximations [7,8]. In fact Elwalid and Mitra [3] derive exact expressions for $P\{X > x\}$ for any

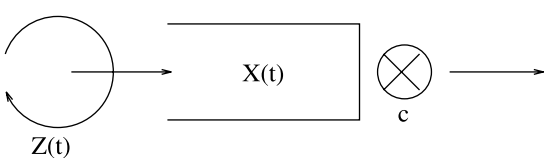


Figure 31.6. Single buffer fluid model.

© 2005 by Chapman & Hall/CRC

CTMC environment $\{Z(t), t \geq 0\}$ process. We use these results and extensions in an example in Section 31.4.2.

To extend the single-node results to a network of nodes, we need two important results. They are summarized as follows:

- *Effective bandwidth of output.* Refer to Figure 31.6. Let $D(t)$ be the total output from the buffer over $(0, t]$. The ALMGF of the output is

$$h_D(v) = \lim_{t\to\infty} \frac{1}{t} \log E\{\exp(vD(t))\}$$

The effective bandwidth of the output traffic from the buffer is

$$\text{eb}_D(v) = \lim_{t\to\infty} \frac{1}{vt} \log E\{\exp(vD(t))\}$$

Let the effective bandwidth of the input traffic be $\text{eb}_A(v)$. Then the effective bandwidth $\text{eb}_D(v)$ of the output can be written as

$$\text{eb}_D(v) = \begin{cases} \text{eb}_A(v) & \text{if } 0 \leq v \leq v^\star \\ c - \frac{v^\star}{v}\{c - \text{eb}_A(v^\star)\} & \text{if } v > v^\star \end{cases}$$

where $v^\star$ is obtained by solving for v in the equation

$$\frac{\mathrm{d}}{\mathrm{d}v}[h_A(v)] = c$$

For more details refer to Chang and co-workers [9,10] and de Veciana *et al.* [11].

- *Multiplexing independent sources.* Consider a single buffer that admits a single-class traffic from K independent sources. Each source k $(k = 1, \ldots, K)$ is driven by a random environment process $\{Z^k(t), t \geq 0\}$. When source k is in state $Z^k(t)$, it generates fluid at rate $r^k(Z^k(t))$ into the buffer. Let $\text{eb}_k(v)$ be the effective bandwidths of source k such that

$$\text{eb}_k(v) = \lim_{t\to\infty} \frac{1}{vt} \log E\{\exp(vA_k(t))\}$$

where

$$A_k(t) = \int_0^t r^k(Z^k(u))\,\mathrm{d}u$$

Let η be the solution to

$$\sum_{k=1}^{K} \text{eb}_k(\eta) = c$$

Notice that the effective bandwidth of independent sources multiplexed together is the sum of the effective bandwidths of the individual sources. The effective bandwidth approximation for large values of x yields

$$P\{X > x\} \approx \mathrm{e}^{-\eta x}$$

© 2005 by Chapman & Hall/CRC

31.4 Case Studies

In this section, case studies are presented to illustrate various performance analysis methodologies as well as to obtain various QoS metrics. The examples are kept simple purely for the purpose of illustration.

31.4.1 Case 1: Delay and Jitter QoS Metrics Using Queueing Networks

Problem 1. *Consider a system of servers arranged as shown in Figure 31.7. Assume that requests arrive according to a Poisson process and enter node 1 at an average rate of 360 per minute. These requests can exit the system from nodes 2, 4, 5, 6, or 7. The processing time for each request in node j (for $j = 1, \ldots, 7$) is exponentially distributed with mean $1/\mu_j$ minutes. The vector of processing rates assume the following numerical values $[\mu_1\ \mu_2\ \mu_3\ \mu_4\ \mu_5\ \mu_6\ \mu_7] = [400\ 200\ 300\ 200\ 200\ 200\ 150]$. There is a single server at each node. When processing is complete at a node, the request leaves through one of the outgoing arcs (assume the arcs are chosen with equal probability). The objective is to obtain the average delay and jitter experienced by all requests. In addition, the average delay and jitter experienced by a particular arriving request that goes through nodes 1–2–5–7 and then exits the network is to be determined. Note that, in this example, jitter is defined as the standard deviation of delay.*

Solution. The system can be modeled as a Jackson network with $N = 7$ nodes or stations. The external arrival rate vector $\theta = [\theta_1\ \theta_2\ \theta_3\ \theta_4\ \theta_5\ \theta_6\ \theta_7]$ is

$$\theta = [360\ 0\ 0\ 0\ 0\ 0\ 0]$$

The routing probabilities are

$$P = \begin{bmatrix} 0 & 1/3 & 1/3 & 1/3 & 0 & 0 & 0 \\ 0 & 0 & 1/3 & 0 & 1/3 & 0 & 0 \\ 0 & 1/4 & 0 & 1/4 & 1/4 & 1/4 & 0 \\ 0 & 0 & 1/3 & 0 & 0 & 1/3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/3 & 1/3 \\ 0 & 0 & 0 & 0 & 1/3 & 0 & 1/3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

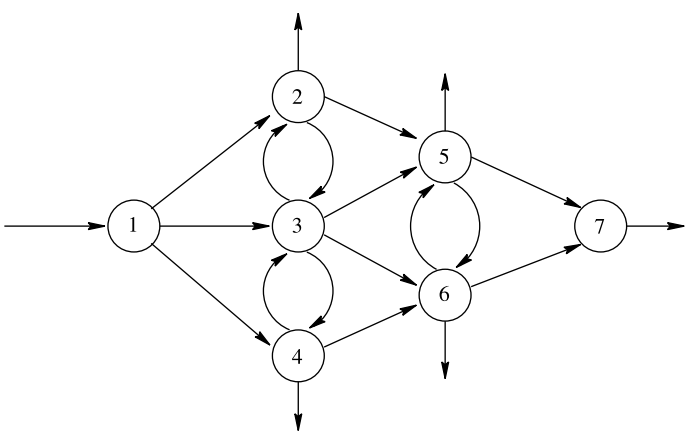


Figure 31.7. Server configuration.

© 2005 by Chapman & Hall/CRC

The effective arrival rate into the seven nodes can be calculated using $\theta(I - P)^{-1}$ as

$$[\lambda_1 \ \lambda_2 \ \lambda_3 \ \lambda_4 \ \lambda_5 \ \lambda_6 \ \lambda_7] = [360 \ 180 \ 240 \ 180 \ 180 \ 180 \ 120]$$

Now, each of the seven nodes can be considered as independent $M/M/1$ queues due to Jackson's theorem. Let L_i be the number of requests in node i in the long run. For $i = 1, 2, \ldots, 7$, the mean and variance of the number of requests in node i are

$$E[L_i] = \frac{\lambda_i}{\mu_i - \lambda_i}$$

and

$$\mathrm{Var}[L_i] = \frac{\lambda_i \mu_i}{(\mu_i - \lambda_i)^2}$$

respectively. Plugging in the numerical values, the average number of pending requests in the seven nodes can be computed as

$$[E[L_1] \ E[L_2] \ E[L_3] \ E[L_4] \ E[L_5] \ E[L_6] \ E[L_7]] = [9 \ 9 \ 4 \ 9 \ 9 \ 9 \ 4]$$

Likewise the variance of the number of pending requests in the seven nodes can be computed as

$$[\mathrm{Var}[L_1] \ \mathrm{Var}[L_2] \ \mathrm{Var}[L_3] \ \mathrm{Var}[L_4] \ \mathrm{Var}[L_5] \ \mathrm{Var}[L_6] \ \mathrm{Var}[L_7]] = [90 \ 90 \ 20 \ 90 \ 90 \ 90 \ 20].$$

Let L be the total number of requests in the system of seven nodes in the long run. Owing to the fact that $L = L_1 + \cdots + L_7$ and independence between nodes, we have

$$E[L] = \sum_{i=1}^{7} E[L_i] = 53$$

and

$$\mathrm{Var}[L] = \sum_{i=1}^{7} \mathrm{Var}[L_i] = 490$$

Let W be the time spent in the network by a request. The delay and jitter, i.e. the performance metrics of interest, are $E[W]$ and $\sqrt{\mathrm{Var}[W]}$ respectively. Using Little's formula and its extensions [1] we have

$$E[W] = \frac{E[L]}{\sum_i \theta_i}$$

and

$$\mathrm{Var}[W] = \frac{\mathrm{Var}[L] + \{E[L]\}^2 - E[L]}{\left(\sum_i \theta_i\right)^2} - \{E[W]\}^2$$

© 2005 by Chapman & Hall/CRC

Therefore, the average delay and jitter experienced by all requests are 0.1472 min and 0.0581 min respectively.

Now, in order to determine the average delay and jitter experienced by a particular arriving request that goes through nodes 1–2–5–7, we use the fact that the time spent by a request in node i is exponentially distributed with parameter $(\mu_i - \lambda_i)$. Therefore, the mean and variance respectively of the time spent in nodes 1, 2, 5 and 7 are [0.0250 0.0500 0.0500 0.0333] and [0.0006 0.0025 0.0025 0.0011]. Since the nodes are independent, the mean and variance of the total times are the sum of those spent at the individual nodes. Therefore, the average delay and jitter experienced by a particular arriving request that goes through nodes 1–2–5–7 are 0.1583 min and 0.0821 min respectively. ■

31.4.2 Case 2: Loss QoS Metrics Using Fluid Models

Problem 2. *Consider a centralized sensor network as shown in Figure 31.8. The six source nodes send sensor data to a sink that processes the data. The intermediary nodes are responsible for multiplexing and forwarding information. The top three sources generate low-priority traffic and the bottom three generate high-priority traffic. Subscript 1 is used for high priority and subscript 0 for low priority. The sensor data traffic can be modeled using on–off sources, such that when the source is on the data are being generated at rate r_i for priority i and when it is off there are no data generated. Let the on and off times be exponentially distributed with parameters α_i and β_i respectively for priority i. Let c_a and c_b be the channel capacities of the buffers, as shown in Figure 31.8. All buffers are of size B. However, assume that the second-stage buffer is partitioned such that a maximum of B_i amount of priority i traffic can be stored in the buffer of size $B = B_0 + B_1$. In addition, at the second-stage buffer, the high-priority traffic is given all available/required processing capacity; anything remaining is given to the low-priority traffic. For this system it is to be determined what loss probability QoS requirements can be met for both priorities. We use the following numerical values: $\alpha_0 = 1$, $\alpha_1 = 2$, $\beta_0 = 0.2$, $\beta_1 = 0.1$, $r_0 = 1$, $r_1 = 2.8$, $c_a = 1.5$, $c_b = 1.35$, $B_0 = 9$, $B_1 = 3$ and $B = B_0 + B_1 = 12$.*

Solution. First, consider the stage 1 buffers. The two buffers are identical in all respects except that the two buffers serve different classes of traffic. The main difference is in the subscript. So we perform the analysis using a subscript i to denote the priority. Let $\text{eb}_{i1}(v)$ be the effective bandwidth of the ith-priority traffic into the corresponding stage 1 buffer. The sources are exponential on–off sources, i.e. they can be modeled as a CTMC. Therefore, we can use the results of Elwalid and Mitra [3] to obtain

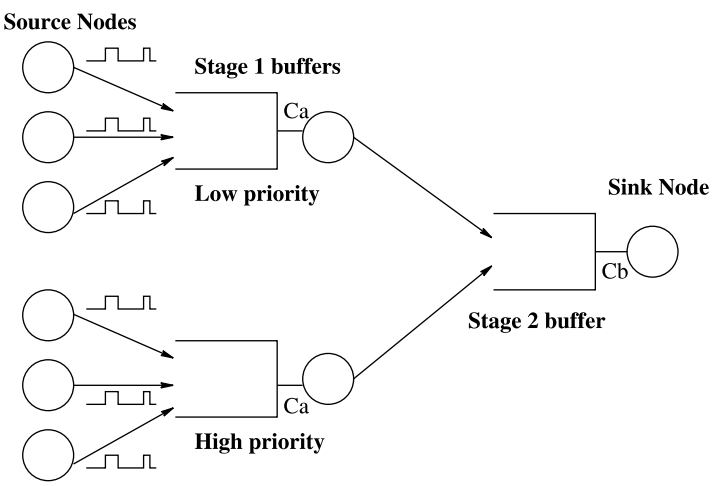


Figure 31.8. Sensor network configuration.

© 2005 by Chapman & Hall/CRC

the effective bandwidth. Since the effective bandwidth of three independent sources multiplexed together is the sum of the effective bandwidths, we have

$$\mathrm{eb}_{i1}(\nu) = \frac{3}{2\nu}\left[r_i\nu - \alpha_i - \beta_i + \sqrt{(r_i\nu - \alpha_i - \beta_i)^2 + 4\beta_i r_i \nu}\right]$$

Further, since the source is a CTMC, we can use exact analysis [12] as opposed to the large deviations result. The loss probability (ℓ_{i1}) for priority i traffic at stage 1 buffer is given by

$$\ell_{i1} = \frac{3\beta_i r_i}{c_\mathrm{a}(\alpha_i + \beta_i)} \mathrm{e}^{-\eta_{i1}B}$$

where η_{i1} is the solution to $\mathrm{eb}_{i1}(\eta_{i1}) = c_\mathrm{a}$, which yields

$$\eta_{i1} = 3\alpha_i/(3r_i - c_\mathrm{a}) - 3\beta_i/c_\mathrm{a}$$

Notice that if we had used large deviations, then ℓ_{i1} would have been just $\mathrm{e}^{-\eta_{i1}B}$ without the constant in front. Plugging in the numerical values (for the exact result, not large deviations), we get $\ell_{01} = 4.59 \times 10^{-9}$ and $\ell_{11} = 3.24 \times 10^{-4}$.

Now consider the stage 2 buffers. In order to differentiate the traffic, we will continue to use subscript i to denote the priority. Let $\mathrm{eb}_{i2}(\nu)$ be the effective bandwidth of the ith-priority traffic into the stage 2 buffer. Using the result for effective bandwidth for the output from buffer of stage 1, we have

$$\mathrm{eb}_{i2}(\nu) = \begin{cases} \mathrm{eb}_{i1}(\nu) & \text{if } 0 \le \nu \le \nu_i^* \\ c_\mathrm{a} - \frac{\nu_i^*}{\nu}\{c_\mathrm{a} - \mathrm{eb}_{i1}(\nu_i^*)\} & \text{if } \nu > \nu_i^* \end{cases}$$

with

$$\nu_i^* = \frac{\beta_i}{r_i}\left[\sqrt{\frac{c_\mathrm{a}\alpha_i}{\beta_i(3r_i - c_\mathrm{a})}} - 1\right] + \frac{\alpha_i}{r_i}\left[1 - \sqrt{\frac{\beta_i(3r_i - c_\mathrm{a})}{c_\mathrm{a}\alpha_i}}\right]$$

For the numerical values above, we get $\nu_0^* = 0.8$ and $\nu_1^* = 0.4105$. The loss probability (ℓ_{i2}) for priority i traffic at stage 2 buffer is given by

$$\ell_{i2} = \mathrm{e}^{-\eta_{i2}B}$$

where η_{12} is the solution to $\mathrm{eb}_{12}(\eta_{12}) = c_\mathrm{b}$ and η_{02} is the solution to $\mathrm{eb}_{02}(\eta_{02}) + \mathrm{eb}_{12}(\eta_{02}) = c_\mathrm{b}$. This is based on the results in Elwalid and Mitra [7] and Kulkarni and Gautam [13]. Plugging in the numerical values, we get $\ell_{02} = 0.0423$ and $\ell_{12} = 0.003$.

Assuming that the loss probabilities are independent across the two stages, the loss QoS requirements that can be satisfied are $1 - (1 - \ell_{01})(1 - \ell_{02})$ and $1 - (1 - \ell_{11})(1 - \ell_{12})$ for priorities 0 and 1 respectively. Therefore, the QoS guarantee that can be provided is that priority 0 and priority 1 traffic will face a loss probability of not more than 4.23% and 0.33% respectively. ■

© 2005 by Chapman & Hall/CRC

31.5 Concluding Remarks

In this chapter, abstract models of networks were studied and how to guarantee QoS for them was analyzed. The work focused on methodologies rather than on applications. The objective was to develop a set of common tools that are applicable to various types of network. In particular, the study of QoS is important and extends to sensor networks. Metrics such as utilization and power consumption could be additional measures that can be derived and be potentially useful. Although it was mentioned earlier in the chapter that issues such as pricing and availability are very crucial for customer satisfaction, we have not paid much attention to them in the case studies on determining QoS metrics. However, while extending the performance analysis to solve design and control problems, it is very important to take pricing and availability into account as well. Another critical aspect that has been left out of this chapter is degradation and failure of resources. A very active research topic that combines issues of availability, QoS, and resource degradation/failure, which in total is called survivability or robustness, has been given a lot of attention recently by both government and industry. Since degradation and failure cannot be modeled quantitatively very well, survivable or robust system designs end up being extremely redundant. Therefore, building cost-effective systems that can be robust or survivable is of utmost importance. Interestingly, sensor networks can be used to address several of the issues!

References

[1] Gross, D. and Harris, C.M., *Fundamentals of Queueing Theory*, 3rd ed., John Wiley and Sons, New York, 1998.

[2] Bolch, G. *et al*., *Queueing Networks and Markov Chains, Modeling and Performance Evaluation with Computer Science Applications.* John Wiley & Sons, 1998.

[3] Elwalid, A.I. and Mitra, D., Effective bandwidth of general Markovian traffic sources and admission control of high-speed networks, *IEEE/ACM Transactions on Networking*, 1(3), 329, 1993.

[4] Kesidis, G. *et al*., Effective bandwidths for multiclass Markov fluids and other ATM sources, *IEEE/ACM Transactions on Networking*, 1(4), 424, 1993.

[5] Kulkarni, V.G., Effective bandwidths for Markov regenerative sources, *Queueing Systems, Theory and Applications*, 24, 137, 1997.

[6] Krishnan, K.R. *et al*., A. Scaling analysis in traffic management of self-similar processes, in *Proceedings of 15th International Teletraffic Congress*, Washington, DC, 1997, 1087.

[7] Elwalid, A.I. and Mitra, D., Analysis, approximations and admission control of a multi-service multiplexing system with priorities, in *INFOCOM'95*, 1995, 463.

[8] Gautam, N. *et al*., Bounds for fluid models driven by semi-Markov inputs, *Probability in the Engineering and Informational Sciences*, 13, 429, 1999.

[9] Chang, C.S. and Thomas, J.A., Effective bandwidth in high-speed digital networks, *IEEE Journal on Selected Areas in Communications*, 13(6), 1091, 1995.

[10] Chang, C.S. and Zajic, T., Effective bandwidths of departure processes from queues with time varying capacities, in *INFOCOM'95*, 1995, 1001.

[11] De Veciana, G. *et al*., Decoupling bandwidths for networks: a decomposition approach to resource management, in *INFOCOM'94*, 1994, 466.

[12] Elwalid, A.I. and Mitra, D., Analysis and design of rate-based congestion control of high speed networks, part I: stochastic fluid models, access regulation, *Queueing Systems*, *Theory and Applications*, 9, 29, 1991.

[13] Kulkarni, V.G. and Gautam, N., Admission control of multi-class traffic with service priorities in high-speed networks, *Queueing Systems, Theory and Applications*, 27, 79, 1997.

© 2005 by Chapman & Hall/CRC

32

Network Daemons for Distributed Sensor Networks

Nageswara S.V. Rao and Qishi Wu

32.1 Introduction

There is a wide spectrum of scenarios in which distributed sensor networks (DSNs) are deployed, ranging from radar sites located across the country that track aircraft, to small robot teams that explore urban areas [1]. Consequently, the networks that underlie DSNs are just as varied, ranging from the long-haul wireline networks to small-area wireless networks. In the former networks, the sustained bandwidth for data transfers and stable channels for control and high-priority traffic are required, while in the latter it is of interest to sustain message delivery under dynamic node movements with very limited or no network infrastructure. Despite the operational diversity, current DSNs are often deployed by utilizing the present Internet technologies, in part due to their wide availability. For example, wide-area DSNs are often connected over the Internet (or networks that are characteristically similar) using hosts equipped with conventional protocol stacks, and wireless DSNs deployed in unstructured areas are often connected using IEEE 802.11 technologies. Consequently, the resultant networks often do not exactly match the DSN requirements, since the Internet technologies are geared toward the best-effort services with the end hosts having very limited control at the network core [2].

The reliance on the Internet technologies by DSNs often manifests in severe performance limitations. For example, in wide-area DSNs, the nodes cannot control routes to avoid the congested regions or accumulate bandwidths over multiple paths (without drastically changing the infrastructure) because the routing at the network core is solely determined by the underlying routers, which are exclusively controlled by the different service providers. Furthermore, there is no support for realizing stable channels for control purpose over such wide-area networks. The throughput achieved for a control channel that employs the most widely used transmission control protocol (TCP) typically underflows under high traffic and overflows under low traffic. Moreover, such a TCP-based control channel may experience very complicated end-to-end transport dynamics in time-varying network conditions. In IEEE 802.11 networks for a team of mobile robots, which, for example, are deployed to assess the radiation levels of a remote area, an infrastructure of access points must be set up prior to the operation [3]. Such a requirement is obviously meaningless if the very goal of the robot team is to assess the

© 2005 by Chapman & Hall/CRC

suitability of the region for human operation. More generally, in DSNs of mobile nodes, the challenges are to form an *ad hoc* wireless network without the infrastructure [4] and to cope with the dynamic changes in network connectivity. Note that the node movements are treated as aberrations in the current Internet environments, where as such movements are shown to improve the message delivery in mobile *ad hoc* networks [5].

The above limitations of wireline and wireless networks are not inherent to DSNs; in fact, the opposite is true: DSNs offer conducive environments for the end hosts and core nodes to cooperate in overcoming several of them. Note that, in principle, networks can be designed from scratch to suit each DSN scenario, but such an approach is too expensive, at least in the short term, since it involves the development of special-purpose hardware and software. We adopt a more pragmatic approach here by utilizing a framework of *network daemons* to enhance the network functionalities to address DSN requirements. These daemons contain modules for measurement, path computation, routing, and transport adaptation, which are all implemented at the application level. These application-level solutions are easily deployable over the current infrastructures and can also be tailored to meet the specific needs of various classes of DSNs.

In this chapter, we consider two specific classes of networks for DSNs: (a) wide-area networks with the requirements of sustained bandwidth and stable control channels, and (b) small-area *ad hoc* wireless networks of mobile nodes deployed in unstructured areas. To address the first class, we employ regression-based path computation [6] and transport stabilization based on a stochastic approximation method [7]. These daemons collect link delay measurements to compute the best paths and use each other to route around high-traffic areas as well as to accumulate bandwidth using multiple paths. For implementing stable control channels, we adopt a source controller that stabilizes a flow using user datagram protocol (UDP). We analytically justify both techniques under fairly general conditions. For the second class of networks, the daemons dynamically track the connectivity changes and exploit the node movements to improve the message delivery [8]; this is a departure from the Internet-based approaches that treat connectivity changes as undesirable.

We present a general network daemon framework in Section 32.2 [9,10], which encompasses link measurement, path computation, transport control, and data routing modules. The analytical and experimental results for the wide-area wireline networks are discussed in Section 32.3. Small-area wireless networks are discussed in Section 32.4. The presentation here is tutorial in nature and the details of various parts can be found elsewhere for wireline networks [6,7,9,10] and for wireless networks [8].

32.2 Network Daemons

The network daemons are deployed at DSN nodes, for example, distributed over either wide-area networks across thousands of miles or *ad hoc* mobile networks confined to small regions. Each daemon consists of four main components, as shown in Figure 32.1:

1. *Link measurement module*. This collects delay or connectivity measurements using test messages that are actively or passively transmitted along virtual links for estimating available link bandwidths and minimum link delays based on a linear regression method or determining the network connectivity.
2. *Path computation module*. This maintains a routing table with link information (either bandwidth or connectivity) from the measured network topology and computes single/multiple paths for the data to be transmitted.
3. *Data routing module*. This routes the data to the destination via the other daemons using path information specified under this framework. The data may be buffered at the intermediate nodes in *ad hoc* mobile networks due to the highly dynamic connectivity.
4. *Transport control module*. This provides the customized data transfer services to meet performance requirements, such as throughput stabilization and connectivity through time, which are specific to a particular DSN.

© 2005 by Chapman & Hall/CRC

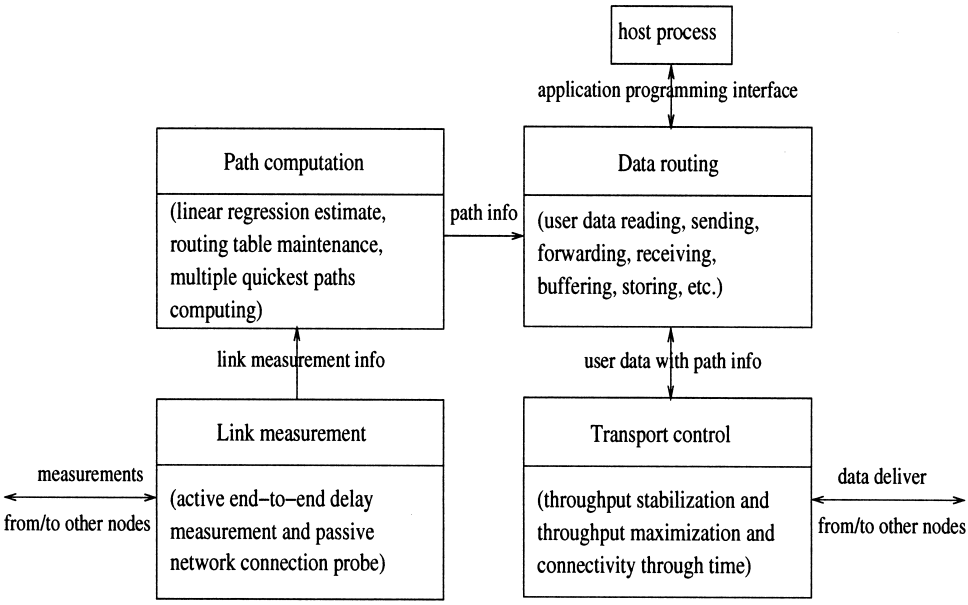


Figure 32.1. Functional block diagram of a daemon.

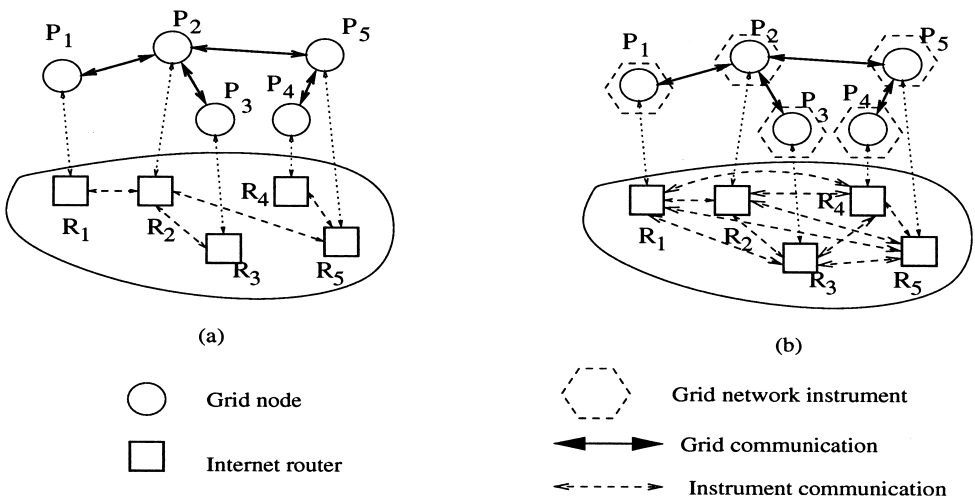


Figure 32.2. Typical scenario of wide-area sensor network over wireline network.

To illustrate the concept of network daemons, consider a computation distributed at several nodes over the Internet, where the messages are communicated as per the task structure shown in Figure 32.2(a). The communication between processes, e.g. between P_1 and P_2, could be handled by a process-to-process TCP stream. The paths traversed by the data packets are decided by the network routers based on a best-effort basis; hence, they are usually not optimized for the host performance. For example, when the background traffic increases on a shared link, a TCP stream experiences longer delays and higher packet losses, thereby curtailing its throughput. The current Internet typically does not allow the hosts to enforce rerouting to avoid highly congested network segments. Although some level of rerouting might be performed by routers, optimizing the end-to-end performance of any particular host process is not their primary goal.

© 2005 by Chapman & Hall/CRC

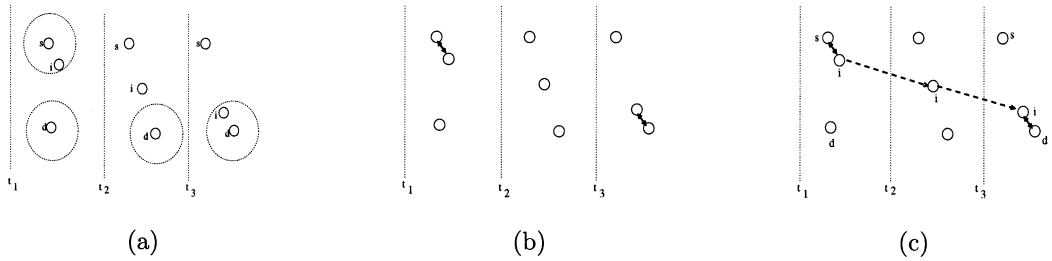


Figure 32.3. Typical scenario of dynamic sensor nodes over wireless network.

Now we consider that network daemons are deployed to assist data transfer as shown in Figure 32.2(b). In one scenario, if link R_1–R_2 is congested but not the other links, then the messages can be sent from R_1 to the routing daemon R_3 then to R_2 via the link R_3–R_2. In another scenario, if the available bandwidth of link R_1–R_2 is not sufficient, then a multiple path consisting of R_1–R_2, R_1–R_3–R_2, R_1–R_4–R_2, and R_1–R_5–R_2 can be utilized. The paths here are implemented using the application-level daemon routers, and each virtual link in such paths may correspond to a number of physical paths via the underlying Internet routers. The decisions about how to choose the appropriate paths may be made based on the measurements collected by the daemons, as will be illustrated in Section 32.3. Since a DSN consists of a number of nodes under a single control, the daemons required can be easily executed on various nodes.

We now consider an *ad hoc* wireless network consisting of three nodes (mobile robots), as shown in Figure 32.3 to illustrate how daemons can be utilized to overcome the reachability limitations of the current methods. Here, there is no backbone network of access points and the nodes communicate with others within a certain range, i.e. only with directly connected neighbors. Consider that a message is to be delivered from robot s to robot d within the time interval $[t_1, t_3]$. In this scenario, the robots s and d are never within the transmission range of each other, which makes the existing protocols inapplicable. TCP cannot deliver the message, since it requires that destination be reachable. The same is true for more special protocols, such as TORA [11], that are designed for mobile networks. We now illustrate that, by using suitable daemons, it is indeed possible to deliver the message. Consider Figure 32.3, which shows the robot movements. Initially, robots s and i are directly connected. Then i moves away from s and all three robots are disconnected during this period. Finally, i moves within the range of d and hence is connected to it. A message sent from s at time t_1 can be sent to i initially where it can be buffered at the routing daemon and then delivered to d at time t_3. Essentially, the movements of i are utilized to deliver the message to d, where the daemons are used to buffer the message in time. More details of this class of applications are discussed in Section 32.4.

32.3 Daemons for Wide-Area Networks

We consider a subclass of DSN networks that are connected over wide-area networks with two types of requirement. First, messages of various sizes must be transported quickly between various nodes, e.g. sensor measurements from end nodes to fusion centers. Second, some of the sensors must be interactively controlled from remote nodes.

32.3.1 Path Computation

For the first task, the variability of message sizes must be accounted for explicitly due to the nonmonotonicity of end-to-end delays: a path with high bandwidth (suited for bulk transfers) is not necessarily suitable for transmitting small messages. Indeed, small messages may be delivered more

© 2005 by Chapman & Hall/CRC

quickly via paths with smaller bandwidth and latency. In the current methods, the messages are typically sent as single or parallel TCP streams, and the protocol stack can be optimized to account for various host parameters [12]. Such host-based methods are very effective for the Internet but do not exploit the physical diversity of paths in the network. Since the network paths are solely determined by the routers on a best-effort basis, it is quite possible that traffic is routed via congested paths while there are other underutilized paths.

The end-to-end delays are also subject to limitations imposed by the queuing policies and traffic loads at the routers, in addition to the bandwidth limits of the links. While the latter delays are somewhat measurable and predictable, those at the routers cannot be very easily modeled. Consequently, the end-to-end delays of messages contain significant random components, whose distributions can be highly complicated [13]. In such cases, the usual formulation of path computing with the least expected end-to-end delay is not viable, since the required distributions are very difficult to estimate. In this section, we adopt a purely measurement-based method wherein the required paths are computed using *in situ* measurements.

An *overlay network of daemons* is represented by a graph $G = (V, E)$ with n nodes and m virtual links. Here, each node represents a daemon and each link represents a communication channel, such as a TCP connection. A message of size r must be transmitted from a source node s to a destination node d, which incurs three types of delay:

1. *Link delay.* For each link $e = (v_1, v_2)$, there is a link-delay $d(e) \geq 0$ such that the leading edge of a message sent via e from node v_1 at time t will arrive at node v_2 at time $t + d(e)$.
2. *Bandwidth-constrained delay.* Each link $e \in E$ has a deterministic "effective" bandwidth $b(e) \geq 0$. Once initiated, a message of r (constant) units can be sent along link e in $r/b(e) + d(e)$ time.
3. A message of size R (random variable) arrives at the source s according to an *unknown* distribution P_R. At any node v, Q_v and R_v are the random variables denoting the queuing delay and message size distributed according to *unknown* distributions $\mathbf{P}_{Q_v}$ and $\mathbf{P}_{R_v}$ respectively. No information about the distributions of R_v and Q_v, $v \in V$, is available. Instead, the *measurements* $(Q_{v;1}, R_{v;1}), (Q_{v;2}, R_{v;2}), \ldots, (Q_{v;l}, R_{v;l})$ that are independently and identically distributed (iid) according to the distribution $\mathbf{P}_{Q_v, R_v}$ are known at each node $v \in V$.

According to the above link model, the bandwidth and minimum link delay of a virtual link in the overlay network can be estimated through active measurements using the following steps:

Step 1. The source node generates a set of test messages of various sizes.

Step 2. The source node divides each message into a number of components of a certain read/send buffer size and transmits them to the destination node through a TCP channel. Note that, internally, all message components are chunked into segments of minimum segment size (MSS) at the TCP layer, each of which is probably further fragmented into data packets at the Internet protocol (IP) layer, depending on the underlying link minimum transfer unit (MTU).

Step 3. The destination node receives message components and acknowledges to the source node the completion of transmission.

Step 4. Upon the receipt of acknowledgments, the source node calculates the end-to-end message delays and applies a linear regression to fit the measured points of message size and end-to-end delay pair. The first-order approximation of the available bandwidth and the minimum link delay are then estimated by the slope and intercept of the regression line respectively.

Such link measurement examples are shown in Figure 32.4, where we consider messages with widely ranging sizes transmitted between Oak Ridge National Laboratory (ORNL) and a number of universities. In the left figure, each cluster corresponds to a single destination. For each destination, each plot corresponds to the measurements collected within the span of a few minutes by randomly picking the message sizes and sending them to the destination. In the right figure we show the delays

© 2005 by Chapman & Hall/CRC

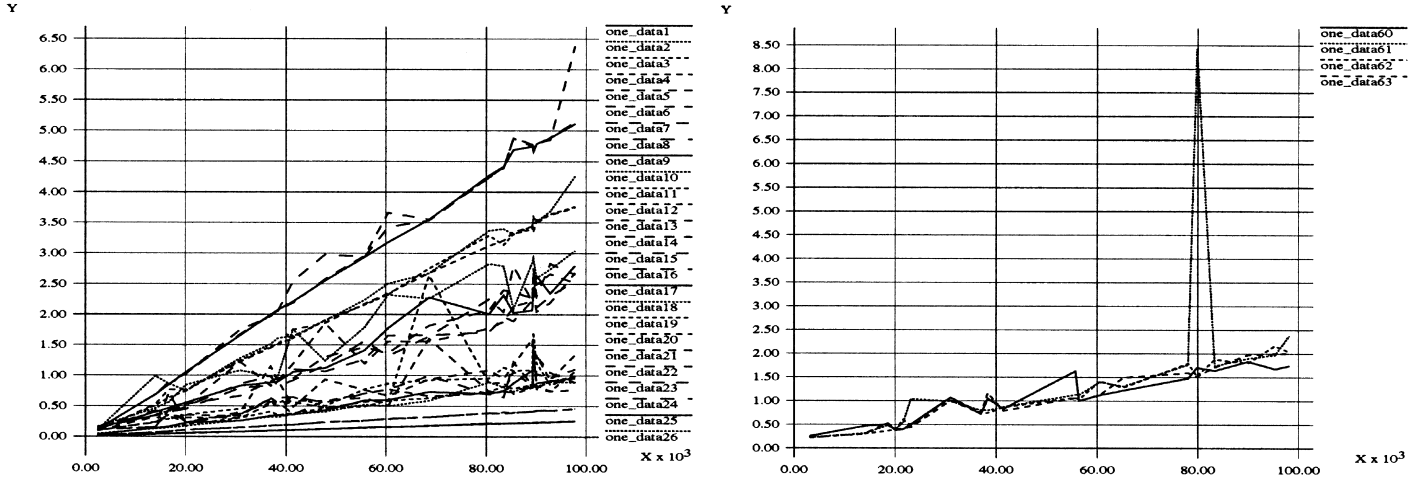


Figure 32.4. ORNL–OU: end-to-end delays for large messages. *X*-axis: message size in bytes; *Y*-axis: end-to-end delay in seconds.

© 2005 by Chapman & Hall/CRC

between ORNL and the University of Oklahoma (OU). As portrayed by these measurements, our model captures the essence of end-to-end delays: in each plot, the "slope" corresponds to the effective link bandwidth and the additional variation corresponds to the random queuing delay Q_v.

32.3.1.1 Probabilistic Delay Guarantees

Consider a path P, from source $s = v_0$ to destination $d = v_k$, given by (v_0, v_1), (v_1, v_2), $\ldots$, (v_{k-1}, v_k), where $(v_j, v_{j+1}) \in E$, for $j = 0, 1, \ldots, (k-1)$. The bandwidth of this path is $b(P) = \min_{j=0}^{k-1} b(e_j)$, and the delay due to bandwidth is $r/b(P)$. The link delay of this path is $d(P) = \sum_{j=0}^{k-1} D(e_j)$. The end-to-end delay of path P in transmitting a message of size R is the sum of these three delay components:

$$T(P, R) = \frac{R}{b(P)} + d(P) + \sum_{j=0}^{k-1} Q_{v_j|R} \tag{32.1}$$

where $Q_{v_j|R}$ is the conditional queuing delay at node v_j given that a message of size R arrived at the node. The *expected* end-to-end delay of path P for the given message size R is given by

$$\bar{T}(P, R) = \frac{R}{b(p)} + d(P) + \sum_{j=0}^{k-1} \int Q_{v_j} \, \mathrm{d}P_{Q_{v_j}|R} \tag{32.2}$$

which is a random variable (of R) for a fixed path P. Let $\mathcal{P}$ denote the set of all paths from s to d. Let P_R^* denote a path with the minimum expected end-to-end delay for the given message size R such that $\bar{T}(P_R^*, R) = \min_{P \in \mathcal{P}} \bar{T}(P, R)$. If the error distributions are known, then P_R^* can be computed using deterministic optimization methods. Such an approach is infeasible here since, in practice, the error distributions of queuing delays are so complicated that they are essentially unknown. We compute an estimator $\hat{P}_R$ of P_R^* using a regression estimator such that

$$\mathbf{P}\left\{\mathbf{E}_R[\bar{T}(\hat{P}_R, R) - \bar{T}(P_R^*, R)] \geq \epsilon\right\} \leq \delta \tag{32.3}$$

for a sufficiently large sample size, which depends on ϵ, δ, n, and a suitably chosen function family $\mathcal{Q}_v$ that contains the regression function. Informally, this condition guarantees that: *the expected delay of $\hat{P}_R$ is within ϵ of that of P_R^* with probability $1 - \delta$, irrespective of the delay distributions.*

This is the best guarantee possible using a measurement-based approach, whose derivation details is given by Rao [6]. Often in networks, measurements are collected to estimate the distributions, which are then used to compute best paths. In the present problem, such an approach can only result in guarantees, which are strictly weaker than that defined in Equation (32.3) mainly because Q_v can have an arbitrary distribution. Informally speaking, the estimation of distributions involves an infinite dimensional quantity, whereas the computation of $\hat{P}_R$ involves minimization over the finite set $\mathcal{P}$.

This guarantee is possible because the measurements are the actual delays collected by the daemons. Traditionally, Internet control message protocol (ICMP)-based mechanisms, such as ping and traceroute, are used to collect measurements. However, the end-to-end guarantees in Equation (32.3) cannot be provided based on such data because some firewalls disable responses to ping and traceroute and sometimes even deliberately send incorrect responses. Also, some firewalls enforce rate controls on ICMP traffic but not on TCP, in which case the delay measurements collected through ping and traceroute could be highly misleading. In a certain sense, our approach not only provides analytical guarantees but also provides guidance for the appropriate measurements.

An algorithm is available [6] to compute the best empirical path $\hat{P}_R$ based on a regression estimator. The complexity of this algorithm is $O(m^2 + mn \log n + nf(l))$, where $f(l)$ is the complexity of computing the regression at a given value r. Thus, a polynomial-time (in l) regression estimator results in a polynomial-time (both in n and l) path computation method.

© 2005 by Chapman & Hall/CRC

32.3.1.2 Multiple Path Computation

A multiple path from s to d, denoted MP, consists of a set of simple bandwidth-disjoint paths from s to d. For simplicity, consider a network $G = (V, E)$ with zero queuing delays $Q_v = 0$ for all $v \in V$ such that a message of r units can be sent along the edge P in $r/B(P) + D(P)$ time. The *end-to-end delay*, denoted by T (MP, r), of a multiple path MP from s to d is defined as the time required to send a message of size r from s to d, wherein the message is subdivided and transmitted via the constituent paths. The multiple paths often provide more bandwidth than a single path if the message can be suitably divided into parts.

Consider a network consisting of two paths P_1 and P_2 such that $B_1 = 10$ units/s, $B_2 = 20$ units/s, $D_1 = 2$ s, and $D_2 = 12$ s. For a message of size $r = 100$ units, $T(P_1, 100) = 12$ and $T(P_2, 100) = 17$. If a single path is used, P_1 will be chosen for this message size. If 99 units are sent on P_1 and 1 unit is sent on P_2, then the corresponding delays are given by $99/10 + 2 = 11.9$ s and $1/20 + 12 = 12.05$ s respectively, resulting in an end-to-end delay of 12.05 s. Hence, two-path $\{P_1, P_2\}$ is not a good choice for this message size. For message of size $r = 1000$ units, the end-to-end delays of single paths P_1 and P_2 are calculated as 102 s and 62 s, respectively. Thus if a single path is used, then P_2 will be chosen. For a two-path $\{P_1, P_2\}$ for $r = 1000$, such that 400 units and 600 units are sent via P_1 and P_2 respectively, this results in individual delays of 42 s for each path. Hence, the resultant end-to-end delay is 42 s, which is smaller than that of P_1 or P_2, which are 102 s and 62 s, respectively.

In general, for two paths P_1 and P_2, we have $T(\{P_1, P_2\}, r) \leq \min\{T(P_1, r), T(P_2, r)\}$ if and only if the condition $C(P_1, P_2)$ given by

$$D(P_1) + \frac{r}{B(P_1)} \geq D(P_2) \quad \text{and} \quad D(P_2) + \frac{r}{B(P_2)} \geq D(P_1)$$

is satisfied. Under this condition, the minimum end-to-end delay of $\{P_1, P_2\}$ is achieved by dividing the message into two parts of sizes r_1 and r_2, $r_1 + r_2 = r$, which are sent via P_1 and P_2 respectively. The sizes of the two parts are given by

$$r_1 = \frac{B_1 r}{B_1 + B_2} + \frac{B_1 B_2 (D_2 - D_1)}{B_1 + B_2} \quad \text{and} \quad r_2 = \frac{B_2 r}{B_1 + B_2} - \frac{B_1 B_2 (D_2 - D_1)}{B_1 + B_2}$$

The general conditions for dividing the message among p paths are given by Rao [6].

The path computation daemon computes the constituent paths of a multiple path for a given message size by repeatedly computing the quickest path and removing it from the graph by reducing the appropriate bandwidths of the links. Then, the extracted quickest paths are combined as per the above conditions. Note that the resultant multiple path is not always guaranteed to be optimal in a strict sense because of the additional unaccounted randomness in the delays. But such paths yielded very good results in actual implementations, as shown in the next section.

32.3.1.3 Internet Implementation

A distributed computing environment of four sites is shown in Figure 32.5, which is used in our implementation. The server is located at OU and the client is located at ORNL. The daemons are implemented using socket programming in C++ under a Linux/UNIX operating system. The delay regression estimation is based on the potential function method [6]. Daemons are executed at the sever and client, and at two additional locations at Louisiana State University (LSU) and Old Dominion University (ODU). Typical experimental results are shown in Figure 32.5 on the right, where the upper curve represents the delays in a single TCP stream ORNL–OU plotted as a function of randomly chosen message sizes. The lower curve corresponds to the multiple path consisting of TCP streams ORNL–ODU–OU, ORNL–LSU–OU and two direct parallel TCP streams ORNL–OU.

© 2005 by Chapman & Hall/CRC

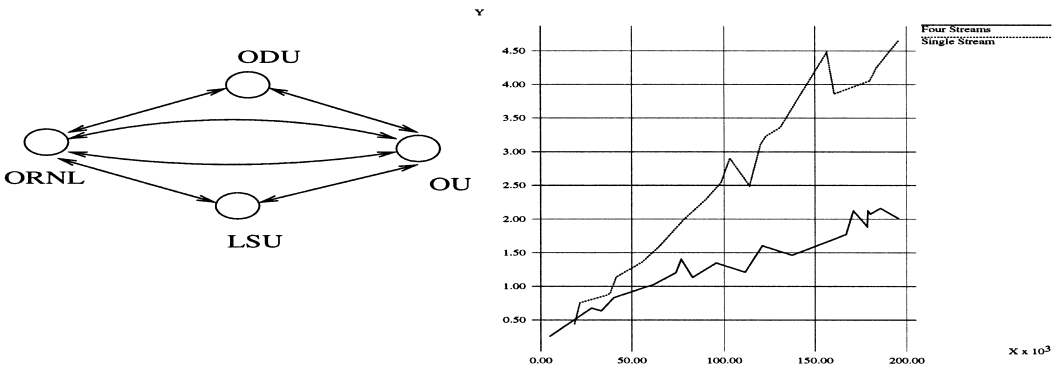


Figure 32.5. ORNL–OU: end-to-end delays for large messages. *X*-axis: message size in bytes; *Y*-axis: end-to-end delay in seconds.

All the overheads of the daemons, namely the path computation and routing times, are included in the measured end-to-end delays. The messages are divided into four parts, as per the delay curves of the paths, and are sent along the respective paths. The overall end-to-end delay when daemons are employed is much lower than the delay with a single TCP stream, except for some smaller sizes. The multiple path resulted in an average improvement of about 35% in the end-to-end delays in the cases we studied. The details of the experimentation and more extensive measurements are given by Rao [6].

32.3.2 Transport Control for Throughput Stabilization

We now address the second task of implementing a control channel over wide-area networks. The bandwidth of the control channel is typically only a small fraction of the available bandwidth, but it is extremely important that the throughput rate at remote site(s) be stable in the presence of dynamic traffic conditions. Large amounts of jitter in throughput can destabilize the control loops needed for remote robots, possibly causing severe damage to them. TCP is not designed for providing such stable throughput. It always continues to increase its throughput until losses are encountered, which often results in much higher throughput than needed. Furthermore, it drastically reduces throughput in response to bursty losses, which might result in long delays in control messages. In general, the nonlinear TCP dynamics make throughput stabilization very challenging [14].

In this section we describe transport daemons based on the stochastic approximation method, which achieves provably stable throughput under very general conditions. Consider stabilizing a transport stream from a source node S to a destination node D over a wide-area network, typically the Internet. The objective is to achieve a *target* throughput rate τ at D by dynamically adjusting the sending rate $r_S(t)$ at S in response to network conditions. Packets are sent from the transport daemon at S and are acknowledged by the daemon at D. Both the original packets and their acknowledgments can be delayed or lost altogether during the transmission due to a variety of reasons, including buffer occupancy levels at routers and hosts and link-level losses. Let $r_S(t)$ and $t_D(t)$ denote the *sending rate* at S and *throughput* or *goodput* at D respectively. The *response plot* corresponds to values of $t_D(.)$ plotted against $r_S(.)$.

In practice, one only has access to various measurements at the source (including the ones that are sent by destination) that need to be utilized to adjust $r_S(t)$. Consider the measurements collected over the Internet shown in Figure 32.6 between ORNL and LSU. In the horizontal plane each point corresponds to a window size and waiting time (or idle time) pair, the ratio of which specifies $r_S(t)$; the top and bottom plots represent $t_D(t)$ and loss rate respectively. For illustration purposes, let us fix the

© 2005 by Chapman & Hall/CRC

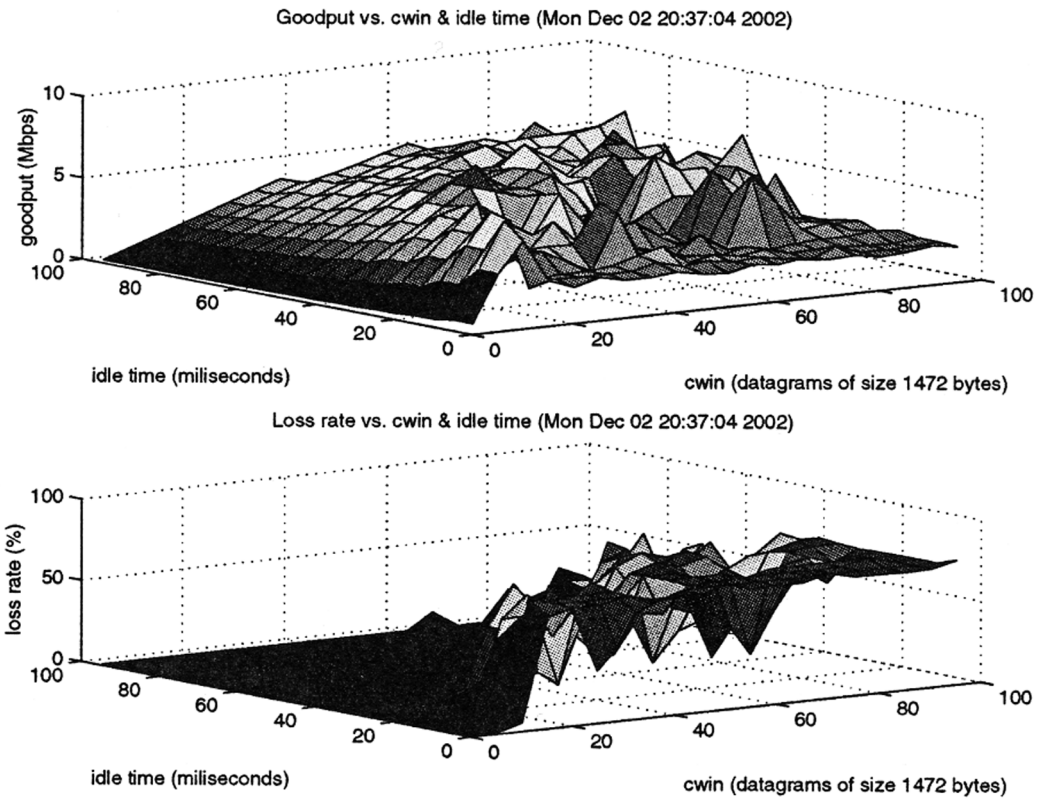


Figure 32.6. Internet measurements with sending rate along horizontal plane. Top: throughput along vertical axis; bottom: loss rate along vertical axis.

waiting time and increase the window size, which corresponds to taking vertical slices of the plots parallel to the window-size axis. There are three important features. (1) There is an overall trend of increase followed by decrease in t_D as r_S is increased; this overall behavior is quite stable, although the transition points vary over time. (2) The plot is quite nonsmooth, mostly because of the randomness involved in packet delays and losses; derivation of smooth utility functions from the response plots is inherently approximate and requires a large number of observations in a preprocessing stage. (3) In practice, S has only an approximation $\hat{t}_S(t)$ to $t_D(t)$, typically computed based on acknowledgments.

We assume that the target throughput τ is specified much below the peak of the response plot. Let $\mathcal{T}(r)$ be the *response regression* given by the expected value of t_D corresponding to fixed $r_S(t) = r$, i.e.

$$E[t_D(t)|r_S(t) = r] = \mathcal{T}(r)$$

Let the *stabilization rate* r_τ be given by $\mathcal{T}(r_\tau) = \tau$. Here, τ is chosen such that r_τ is within the initial increasing part of $\mathcal{T}(.)$. We assume that $\mathcal{T}(r)$ is locally monotonic in the neighborhood of τ, such that $\mathcal{T}(r) > \tau$ for $r > r_\tau$ and $\mathcal{T}(r) < \tau$ for $r < r_\tau$. This assumption is consistent with the measurements in Figure 32.6 and also with the concavity assumptions of Kelly [15] and Low *et al.* [16]. Note that $t_D(t)$ can be highly nonsmooth and $\mathcal{T}(.)$ is not known.

© 2005 by Chapman & Hall/CRC

32.3.2.1 Throughput Stabilization

Our method is based on a simple flow control mechanism at S, where $r_S(t)$ is adjusted in response to a dynamically computed estimate $\hat{t}_S(t)$ of $t_D(t)$ based on acknowledgments received from D. At time t_i, $W(t_i)$ denotes the number of packets to be sent followed by a waiting time $T(t_i)$ such that

$$r_S(t) = \frac{W(t)}{t_a + T(t)}$$

for $t_i \le t < t_{i+1}$, where t_a is time needed for transmitting the packets at S and t_{i+1} is the time when $W(.)$ is updated next. We fix $T(t) = T_0$ and update the window size as follows:

$$W(t_{i+1}) = W(t_i) - \gamma_i[\hat{t}_S(t_i) - \tau] \tag{32.4}$$

where $\gamma_i = K(t_a + T_0)/i^\alpha$ for $0.5 < \alpha < 1$ and $K > 0$ a suitably chosen constant. This method is a specific form of the well-known stochastic approximation (SA) algorithm [17]. Intuitively, this algorithm increases $W(.)$ if the estimate of the throughput is below τ and decreases it otherwise. Initially, at $t = t_0$, T_0 and $W(t_0)$ are chosen based on our initial measurements so that $r_S(t_0)$ is within the vicinity of τ.

Let $W_\tau = r_\tau(t_a + T_0)$ correspond to the ideal window size that achieves the stabilization rate τ in an expected sense. We assume that $\mathrm{Var}[\hat{W}_S(t_{i+1})|W(t_1), W(t_2), \ldots, W(t_i)] \le \sigma^2$ for some σ, and there exist K_0 and K_1 such that

$$K_0|r - r_\tau| \le |\mathcal{T}(r) - \tau| \le K_1|r - r_\tau|$$

Under these conditions, we have the stability result [7]: $E[(W(t_i) - W_\tau)^2] = O(i^{-\alpha})$. By taking into account the scale factor, we have $E[(r_S(t_i) - r_\tau)^2] = O(i^{-\alpha})$. This result is valid even when r_τ varies over time but somewhat slowly. Since $\hat{t}_S(t)$ is a noisy estimate of a random quantity, it is very critical that the step size γ_i in Equation (32.4) be chosen to satisfy the classical Robbins–Monro conditions [18]: (i) $\gamma_i \to 0$ as $i \to \infty$; (ii) $\sum_{i=1}^{\infty} \gamma_i = \infty$. The above algorithm and its stability analysis can be repeated for the case $W(t) = W_0$ is fixed and $T(t)$ is changed in a manner similar to Equation (32.4). Our experimental results are qualitatively identical in both cases.

32.3.2.2 Experimental Results

Our method is tested extensively between ORNL and LSU. During the testing, ORNL is connected to ESnet, which peers with Abilene network in New York. Abilene runs from New York via Washington, DC, and Atlanta to Houston, where it connects to LSU via a regional network. In terms of network distance, these two sites are separated by more than 2000 miles, and both ESnet and Abilene have significant traffic. Figure 32.7 shows typical results for target throughput at 2.5 Mbps which is below the peak bandwidth but above throughput 1.09 Mbps achieved by default TCP. In each plot, the top and bottom curves correspond to $r_S(t)$ and $\hat{t}_S(t)$ respectively, which often overlap indicating low loss conditions. The stabilization typically occurred under very low, albeit nonzero, packet loss. The throughput was remarkably robust and was virtually unchanged when transfers of various file sizes using ftp were made at local and other LAN hosts together with various Web browsing operations.

32.4 Daemons for *Ad Hoc* Mobile Networks

The DSNs of mobile nodes can be applied in a wide variety of scenarios. For example, a robot team can be deployed (perhaps air-dropped) to build a radiation map of an urban area suspected of nuclear or

© 2005 by Chapman & Hall/CRC

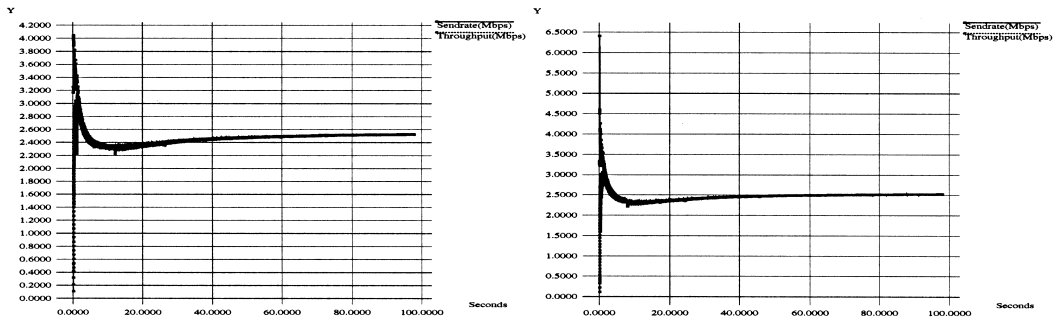


Figure 32.7. Stabilization at 2.5 Mbps throughput under various background traffic. Left: large file ftp at host with Web browsing; right: large file ftp from different local-area network (LAN) nodes with Web browsing.

chemical contamination before human operators are allowed into the area [19]. Typically, in these applications there is a need for the robots to communicate effectively to coordinate their activities, as well as to combine the information gathered. The networking needs for this class of applications are quite specific and are not adequately addressed by the existing wireless *ad hoc* networking technologies.

In general, various types of scenarios call for different types of DSN wireless network [4]. In *ad hoc* wireless networks, the challenge is to form and operate a network without the infrastructure. In dynamic networks, the additional challenge is to cope with the changes in network connectivity. Several network protocols have been developed for various sensor network scenarios (see [20,21] and references therein). The specific class of wireless *ad hoc* networks discussed above leads to the following considerations.

- *Small number of nodes.* We consider networks of tens of nodes which cooperatively perform a task. Its primary focus is to execute a cooperative mission, and the node movements are not tasked exclusively for communication purposes.
- *No infrastructure.* The sensor nodes operate over a wireless network in areas that are typical indoor or urban environments. The radio connectivity is highly dynamic and unpredictable due to the unstructured nature of the terrain and node movements.
- *No special hardware.* We consider that the sensor nodes are equipped with IEEE 802.11 wireless cards, and no special communication hardware is available.

In existing wireless sensor networks it is common to employ Internet wireless network technologies, typically IEEE 802.11 wireless cards and a default TCP/IP stack. In the default infrastructure mode, nodes communicate exclusively through the access points, which requires a backbone of the access points to connect various nodes. The 802.11 cards can be operated in the *ad hoc* mode, in which case the nodes that are within the radio range can communicate with each another but their connectivity is restricted to pairs that are within the radio range. In the Internet-based technologies, the connectivity changes are treated as aberrations and are handled as exceptions. On the other hand, in the above scenarios, the connectivity changes are integral parts of the operation rather than exceptions. More importantly, if suitable protocols are employed, then the connectivity changes can actually improve the network throughput, as shown analytically by Grossglauser and Tse. In this section, we show that the connectivity-through-time concept provides a way to conceptualize such phenomena and to design protocols to exploit the node movements [8].

32.4.1 Connectivity-Through-Time Concept

The graph $G(t) = (V, E(t))$ represents the connectivity of the network at time t with node $v \in V$ representing a robot and edge $(u, v) \in E(t)$ representing a direct wireless communication link between nodes u and v. At time t a path from nodes s to d in $G(t)$ represents a multi-hop network connection,

© 2005 by Chapman & Hall/CRC

since a message can be routed along nodes of the path. If this path persists for a time interval $[T_1, T_2]$ then a message with the end-to-end delay of $T_2 - T_1$ can be successfully delivered from s to d. On the other hand, if there is no path from s to d in $G(t)$ for any t, then it does not necessarily mean that a message cannot be delivered.

To discuss the performance of a protocol that achieves such delivery we need to identify a reasonable performance criterion. Since the topology is dynamic, it is too weak to expect that a datagram be delivered from s to d only if they are connected at some time (e.g. as is done in TORA [11]). On the other hand, it is unreasonable to expect messages to wait indefinitely long in the network; if d is not reachable from s at all, then flooding the messages could lead to inordinate amounts of datagrams being generated, thereby causing the denial of service between the nodes that are connected. To address the issue, the concept of *connectivity-through-time* was proposed [22].

Let the topology changes occur at unique times, denoted in increasing order by $t_1, t_2, \ldots, t_k$ for $t_i \in [0, T]$. Note that $G(t)$ remains constant for all $t \in [t_i, t_{i+1})$ and is given by $G(t_i)$. We define that s and d are 0-*connected-through-time* for interval $[T_L, T_H]$ if they are connected in $G(t)$ for some $t \in [T_L, T_H]$. Consider $[T_L, T_H] \subset (t_{i-1}, t_{i+1})$ containing t_i. We define that s and d are 1-*connected-through-time* for interval $[T_L, T_H]$ if:

1. They are 0-connected through-time for $[T_L, T_H]$.
2. There exists a node v such that (i) s and v are connected in $G(t_i)$, and (ii) v and d are connected in $G(t_{i+1})$.

The *time-path* in $[T_L, T_H]$ is represented by the composition of path from s to v in $G(t_i)$, followed by *time-edge* $(v; t_i, v; t_{i+1})$, and followed by path from v to d in $G(t_{i+1})$. The time interval corresponding to $T_H - T_L$ is called the *hold-time* of the path. This definition is recursively applied to an interval containing more than one t_i as follows. We define that s and d are k-*connected-through-time* for interval $[T_L, T_H]$ containing $t_1, t_2, \ldots, t_k$ if they are:

1. 1-connected-through-time for $[T_L, t_1)$
2. $(k-1)$-connected-through-time for $[t_1, T_H]$

Then s and d are *connected-through-time* for interval $[T_L, T_H]$ if they are k-connected-through-time. We consider that each node v is connected to itself at all times through time-edges denoted by $(v; t_i, v; t_{i+1})$.

One can visualize a time-expanded graph EG($[0, T]$)=(EV, EE) of $G(t)$ as follows. For each interval $[t_i, t_{i+1})$: (i) each $v \in V$ of $G(t_i)$ corresponds to node $v; t_i$ in EV; and (ii) each edge $(u, v) \in E(t_i)$ of $G(t_i)$ is represented by the edge $(u; t_i, v; t_{i+1})$ in EG($[0, T]$). Additionally, for a node v of V, we place the time edge $(v; t_i, v; t_{i+1})$ for each interval $[t_i, t_{i+1}]$. We define a time path from s to d in the expanded graph as a path with the condition that time intervals of all time edges be (a) disjoint and (b) their beginning times be strictly increasing as we move along the path from s to d. Thus a time path typically consists of the usual graph paths in $G(t_i)$ interconnected by time edges. In Figure 32.3(c), the time path is denoted by $(s; t_1, i; t_1)$, $(i; t_1, i; t_2)$, $(i; t_2, i; t_3)$, $(i; t_3, d; t_3)$. The hold time of a time path is the sum of the hold times of all its time edges. Intuitively speaking, if s and d are connected-through-time in $[0, T]$, then a datagram from s can be delivered to d by transmitting along graph paths and buffering along the time edges for a time period given by the hold time.

There are two practical considerations in implementing the above approach. First, the nodes have finite buffers and packets cannot be indefinitely stored. Second, the transmission time is nonzero and could be significant for newly made connections. As a result, not all messages in the buffers may be delivered during the time a connection is available. We parameterize the packets delivery along the connectivity-through-time with two parameters, *time-to-live* and *minimum-connection time*. The first parameter specifies the time during which the current message is useful. For example, the location information of a moving robot is obsolete after a certain time. So we delete the messages from the buffers after the expiry of their time-to-live values. Then packets with appropriate time-to-live value can

© 2005 by Chapman & Hall/CRC

be delivered along a time path with sufficient minimum-connection time. Note that the minimum-connection time depends on the robot movements and time-to-live is a protocol parameter.

32.4.2 CTIME Protocol

The overall idea of this protocol is to track the connectivity and route the packets by suitably buffering them if there is no path to the destination. Each network node acts as a router in delivering the messages. The source nodes decompose the messages as UDP datagrams and send them over the network. The received datagrams are reassembled at the destination. This protocol is specified by two variables, *time-to-live* and *minimum-connection time*, both determined empirically. Each packet is given the same time-to-live value. The minimum-connection time is assumed to be sufficient to clear the buffered packets.

The CTIME protocol is implemented using daemons, as shown in Figure 32.8:

1. *Connectivity computation.* The direct and multiple-hop connectivity at each node is continually updated in response to the I-am-here datagrams broadcast periodically by each node.
2. *Message routing.* The messages are decomposed into UDP datagrams, which are then routed by suitably buffering when needed.
3. *Message transport.* Packet losses, throughput rates, and duplicates are handled by using a window-based mechanism akin to TCP but adapted to the current environment.

32.4.2.1 Connectivity Computation

Each node v maintains a list of *direct neighbors* DN(v) that it is in direct contact with and a list of *multiple-hop neighbors* MN(v) that it is connected with via other nodes. Each node periodically broadcasts an *I-am-here* UDP broadcast packet, which is heard by all nodes within the direct range. This packet includes the list of all direct neighbors and multiple-hop neighbors. Using these messages from the neighbors, each node computes its direct and multi-hop neighbors using the distributed transitive closure algorithm. This information is periodically updated as the I-am-here messages from other nodes are received. The counting to infinity problem is avoided by not sending the connectivity information to its original source, as is usually done.

32.4.2.2 Routing

The message at a source node is decomposed into fixed-sized datagrams, which are routed along the nodes. The datagrams are written to the local routing daemon. The routing module also receives packets from other nodes to be routed or buffered. The packets targeted to the local node are simply sent to the send/receive module. For other datagrams, if a route to destination exits, then it is sent along the known

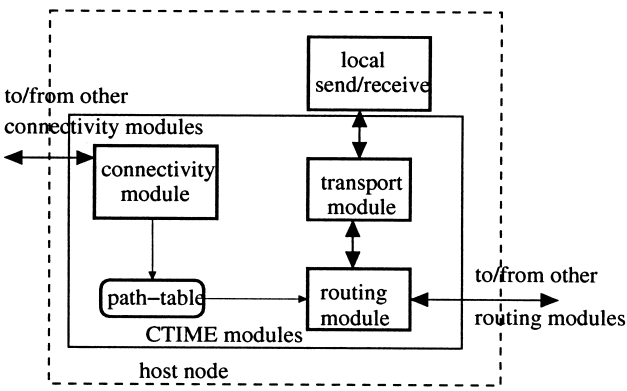


Figure 32.8. Daemons for wireless *ad hoc* networks.

© 2005 by Chapman & Hall/CRC

path by writing it to the next node on the path to destination. That is, if the destination is directly connected, then it is sent to it; if not, then it is written to the next node on the path to destination. If no path to the destination exists, then it is buffered locally if not already buffered and then is broadcast to all its immediate neighbors. When new connections are made, this module examines the list of buffered packets and routes them as above. Also, the buffered packets are periodically examined, and those that outlived their time-to-live values are simply deleted.

32.4.2.3 Transport Method

The transport module at the source generates the UDP datagrams from the message and keeps track of packets that have not been acknowledged. It maintains a buffer of unacknowledged packets and sends them to the router module at the appropriate rates as described below. It also resends the unacknowledged packets after a time-out period. A simple window-based flow control strategy is used at the source and intermediate nodes. Each node maintains a window size w and window time T_w to compute its throughput in terms of the number of packets sent during T_w. The preference of this method over TCP for transport is dictated by the following considerations:

1. *High physical-layer losses.* The usual implementation of congestion control is not suited for this environment due to high packet loss and low probability of simultaneous transmissions at the physical layer. TCP interprets the physical layer losses as congestion signals and reduces its throughput. In the current scenario, however, the opposite is needed: the throughput must be increased to account for the packet loss.
2. *Graceful disconnection.* Owing to high rate of disconnection, the TCP-based method will wait for a connection time-out.
3. *Application-level tuning.* Most TCP parameters are not available to be tuned to suit the application without a kernel modification. For example, it is not easy to select the congestion window size based on the current connection parameters.

At each source we specify the throughput rate depending on the connectivity to the destination. We collected the throughput rates at the source and destination which showed a unimodal behavior (akin to those used in TCP models [16,23]). In the left plot of Figure 32.9 we show the receiver

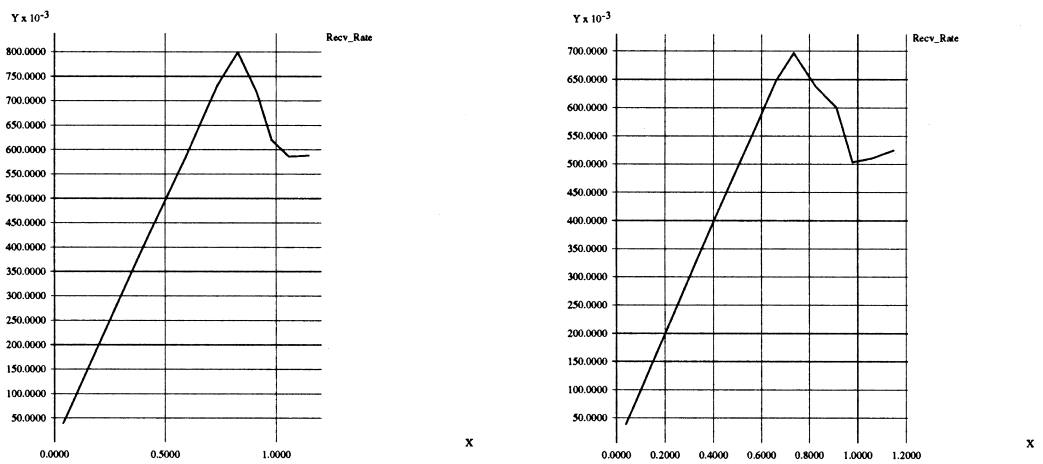


Figure 32.9. Destination throughput (Mbps) versus source sending rate (along *X*-axis in Mbps) for stationary and moving nodes (left and right respectively).

© 2005 by Chapman & Hall/CRC

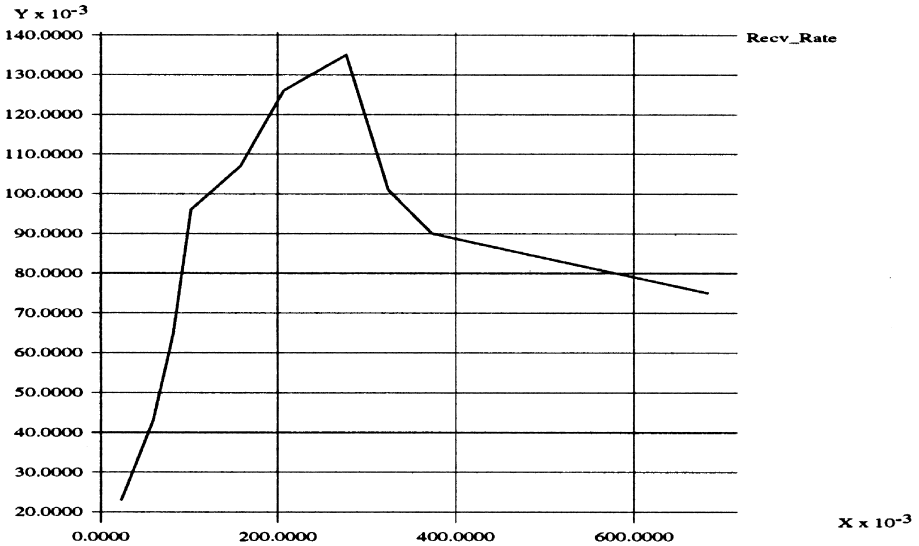


Figure 32.10. Destination throughput (Mbps) versus source sending rate (Mbps) for multiple-hop connections.

throughput as a function of the sending rate when both nodes are stationary. While one or more robots are in motion, there are somewhat higher losses and lower throughput, as shown in the right plot of Figure 32.9. We choose an appropriate sending rate for direct connections based on whether the robots are moving or not.

There is a significant reduction in the overall throughput when packets are routed via other robots, because: (a) same physical channel is used for two connections at the intermediate node, thereby reducing the available raw bandwidth to at most half the peak; (b) the overheads of routing introduce delays, thereby further reducing the bandwidth. Note that TCP is not capable of sending packets using other nodes as routers. Generally, TCP only needs to take care of the transport at the two ends, while the intermediate nodes in the CTIME protocol have much more complicated transport controls to ensure reliable delivery. The throughput at the destination as a function of source sending rate is shown in Figure 32.10 when the packets are routed via an intermediate node. This plot also shows unimodal behavior, but at significantly lower source rate compared with a direct connection. It is important to ensure good throughput to distinguish between the connection from source to destination and employ a suitable sending rate. In CTIME, we apply the direct sending rate either (a) in transmissions to the destination when directly reachable or (b) in broadcasting if the destination is not reachable. If the destination is reachable via multiple hops, then the lower sending rate is employed as per the observations shown in Figure 32.10.

32.4.3 Experimental Results

We present experimental results based on the implementation on a team of mobile robots. The protocol is implemented in C++ under Linux OS using socket-level programming. The testing is carried out on a team of four Mini ATRV mobile robots equipped with 802.11 wireless cards.

In scenario 1, we demonstrate that the robots are used as routers to deliver messages when there is no direct path from source to destination nodes, but the intermediate node falls within the intersection of the radio ranges of source and destination. As shown in Figure 32.11, the intermediate node receives datagrams from the source and forwards them to the destination. Since the connections of two hops exist at all time, the datagrams are continuously transmitted through the path until they arrive

© 2005 by Chapman & Hall/CRC

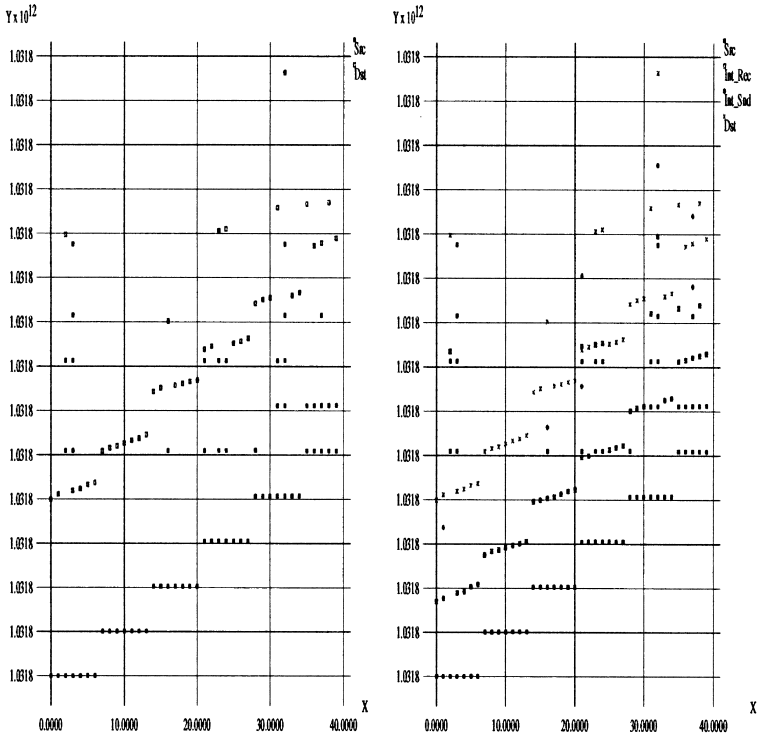


Figure 32.11. Scenario 1: robot serves as a router. Packet number (*X*-axis) versus send/receive time.

at the destination. The corresponding average throughput is calculated and plotted in Figure 32.12. Since the incoming and outgoing connections at the intermediate node contend with each other for the physical channel bandwidth, the intermediate and destination nodes have lower receiving rate than the source and intermediate sending rate.

In scenario 2, we illustrate that messages will be buffered where the path to the destination breaks. As shown in Figure 32.13, the data transmission has three stages. The first stage is similar to scenario 1: the source and destination are connected to the intermediate node but there is no direct connection between them. The intermediate node serves as a router, receiving and forwarding the first set of datagrams. In the second stage, the connection between the intermediate node and destination breaks, so the intermediate node starts buffering incoming datagrams until the connection is brought back up. The second set of datagrams is delivered in the third stage, where the connection of the second hop resumes. Figure 32.14 shows the average throughput calculated for scenario 2. During the second stage, the throughput of the destination decreases because it does not receive any data from the intermediate node, while the intermediate node has a higher sending rate because the preset time-out incurs broadcasting.

In scenario 3, we show that the messages are delivered between source and destination, which are never connected to each other even via multiple hops. As shown in Figure 32.15, this scenario can also be divided into three stages. In the first stage, only the connection between the source and intermediate node exists, and the datagrams are broadcast after a preset timer expires. This connection breaks in the second stage, where the intermediate node is the only active node performing broadcasts. In the third stage, the connection between the intermediate node and destination comes up, so that a new path is found to deliver the datagrams from intermediate node to the destination. As a matter of fact, some of the datagrams are received by the destination through broadcast right after the second hop connection is created and before the new path is computed. The corresponding average throughput is shown in Figure 32.16. Observe from the throughput curves that the destination has almost the same throughput

© 2005 by Chapman & Hall/CRC

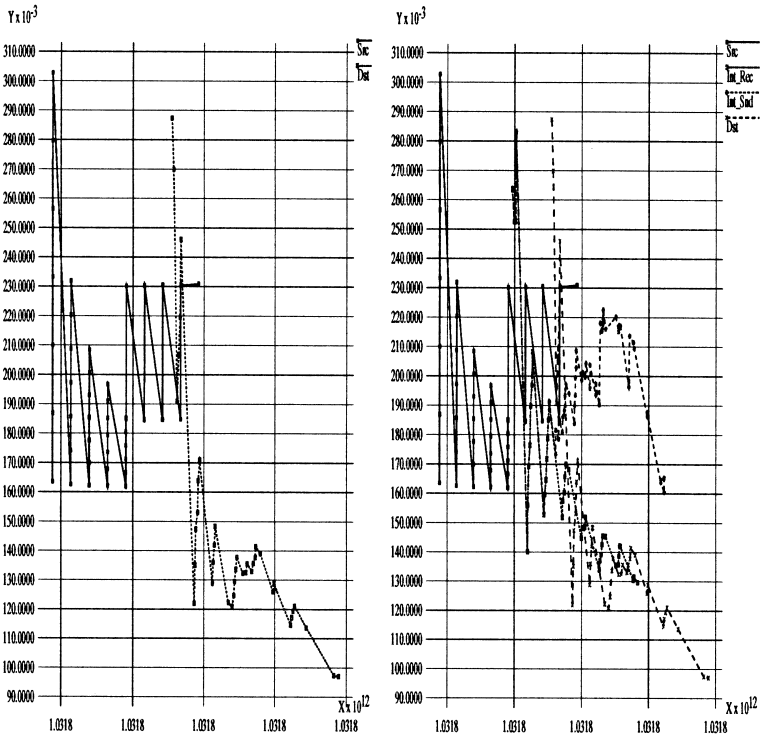


Figure 32.12. Average throughput versus time in scenario 1.

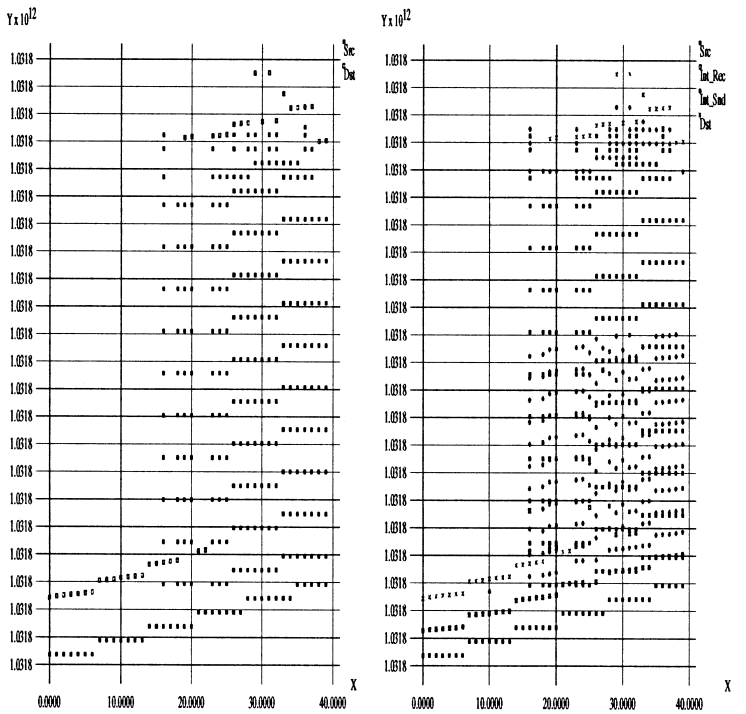


Figure 32.13. Scenario 2: path to the destination breaks. Packet number (*X*-axis) versus send/receive time.

© 2005 by Chapman & Hall/CRC

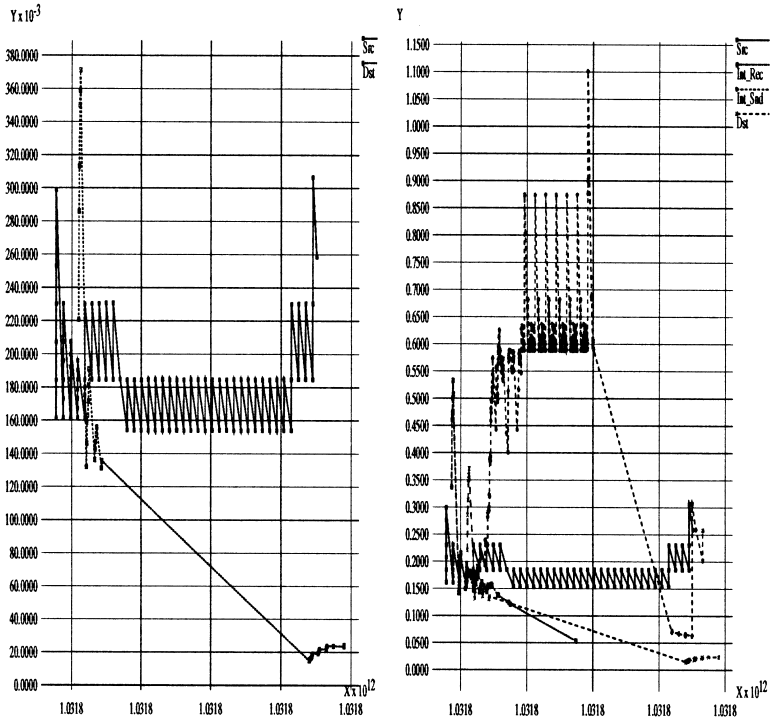


Figure 32.14. Average throughput versus time in scenario 2.

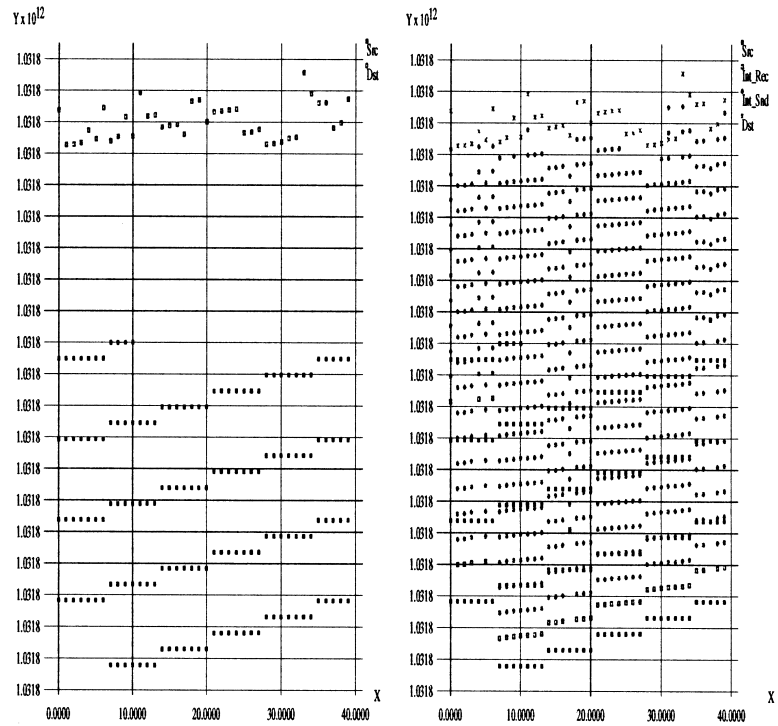


Figure 32.15. Scenario 3: messages are delivered through time-connectivity. Packet number (*X*-axis) versus send/receive time.

© 2005 by Chapman & Hall/CRC

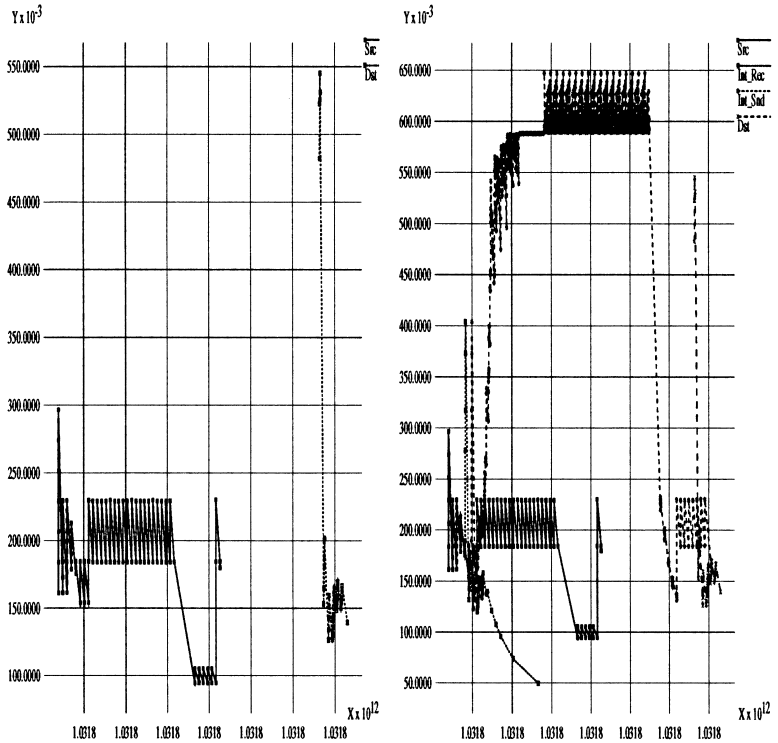


Figure 32.16. Average throughput versus time in scenario 3.

as the intermediate node. The explanation for this observation is that the two hop connections never exist at the same time, so that each of them has the exclusive bandwidth utilization at different times.

32.5 Conclusions

Many DSN deployed in the field utilize technologies developed for the Internet, and hence inherit their limitations. But, they offer operational environments that are fundamentally different from the Internet, wherein application-level daemons can be deployed and customized for various classes of sensor networks. We described a generic framework of network daemons that perform the tasks of link measurement, transport control, data routing, and path computation to overcome several throughput and connectivity limitations. We then described two special classes of daemons for wide-area wireline networks and small-area wireless networks. The first class provided bandwidth aggregation using multiple paths and throughput stabilization using a stochastic approximation method. The second class provided enhanced multi-hop connectivity by exploiting the node movements.

There are several topics to be investigated further. We discussed two very dissimilar classes of DSNs supported by different networks. It would be interesting to see other classes of DSNs to which the generic daemons can be naturally tailored. Another area of interest is to develop daemons that can automatically adapted to the DSN, particularly in cases where the underlying network consists of wireless and wireline networks.

Acknowledgments

This research is sponsored by the Material Science and Engineering Division, Office of Basic Energy Sciences, U.S. Department of Energy, under Contract No. DE-AC05-00OR22725 with UT-Battelle, LLC,

© 2005 by Chapman & Hall/CRC

the Defense Advanced Projects Research Agency under MIPR No. K153, and by National Science Foundation under Grants No. ANI-0229969 and No. ANI-335185.

References

[1] Brooks, R.R. and Iyengar, S.S., *Multi-Sensor Fusion: Fundamentals and Applications with Software*, Prentice-Hall, 1998.
[2] Peterson, L.L. and Davie, B.S., *Computer Networks*, 2nd ed., Morgan Kaufman, 2000.
[3] Gast, M.S., *802.11 Wireless Networks*, O'Reilly, 2002.
[4] Estrin, D. *et al.*, Instrumenting the world with wireless sensor networks, in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, 2001.
[5] Grossglauser, M. and Tse, D., Mobility increases the capacity of *ad-hoc* networks, *IEEE Transactions on Networking*, 10(4), 477, 2002.
[6] Rao, N.S.V., Overlay networks of *in situ* instruments for probabilistic guarantees on message delays in wide-area networks, *IEEE Journal on Selected Areas in Communications*, 22(1), 79, 2004.
[7] Rao, N.S.V. *et al.*, On throughput stabilization of network transport, ORNL manuscript, 2003.
[8] Rao, N.S.V. *et al.*, On throughput stabilization of network transport, in *Proceedings of International Conference on Robotics and Automation*, 2003.
[9] Rao, N.S.V. *et al.*, Netlets: measurement-based routing daemons for low end-to-end delays over networks, *Computer Communications*, 26(8), 834, 2003.
[10] Rao, N.S.V., NetLets for end-to-end delay minimization in distributed computing over Internet using two-paths, *International Journal of High Performance Computing Applications*, 16(3), 285, 2002.
[11] Park, V.C. and Corson, M.S., A performance comparison of the temporally-ordered routing algorithm and ideal link-state routing, in *Proceedings of INFOCOM'98*, 1998.
[12] Web100 concept paper, 1999. http://www.web100.org (last accessed on 8/3/2004).
[13] Willinger, W. and Paxson, V., Where mathematics meets the Internet, *Notices of the American Mathematical Society*, September, 961, 1998.
[14] Rao, N.S.V. and Chua, L.O., On dynamics of network transport protocols, in *Proceedings of Workshop on Signal Processing, Communications, Chaos and Systems*, 2002, 29.
[15] Kelly, F.P., Mathematical modelling of the Internet, in *Proceedings of International Congress on Industrial and Applied Mathematics*, 1999, http://www.statslab.cam.ac.uk/frank/mmi.html (last accessed on 8/3/2004).
[16] Low, S.H. *et al.*, Internet congestion control, *IEEE Control Systems Magazine*, 22(1), 28, 2002.
[17] Kushner, H.J. and Yin, C.G., *Stochastic Approximation Algorithms and Applications*, Springer-Verlag, 1997.
[18] Wasan, M.T., *Stochastic Approximation*, Cambridge University Press, Cambridge, UK, 1969.
[19] Balch, T. and Parker, L.E. (eds), *Robot Teams: From Diversity to Polymorphism*, AK Peter, 2002.
[20] Braginsky, D. and Estrin, D., Rumor routing algorithm for sensor networks, in *First Workshop on Sensor Networks and Applications*, 2002, 22.
[21] Toh, C.K., *Ad Hoc Mobile Wireless Networks*, Prentice-Hall, 2002.
[22] Radhakrishnan, S. *et al.*, DST — a routing protocol for *ad hoc* networks using distributed spanning trees, in *Proceedings of 1999 IEEE Wireless Communications and Networking Conference*, 1999.
[23] Kelley, F.P., Mathematical modelling of the Internet, in *Proceedings of 4th International Congress of Industrial Applied Mathematics*, 1999.

© 2005 by Chapman & Hall/CRC

V

Power Management

33. Designing Energy-Aware Sensor Systems *N. Vijaykrishnan, M.J. Irwin, M. Kandemir, L. Li, G. Chen, and B. Kang* *653*
Introduction • Sources of Power Consumption • Power Optimizations: Different Stages of System Design • Energy Reduction Techniques • Conclusions

34. Operating System Power Management *Vishnu Swaminathan and Krishnendu Chakrabarty* *667*
Introduction • Node-Level Processor-Oriented Energy Management • Node-Level I/O-Device-Oriented Energy Management • Conclusions

35. An Energy-Aware Approach for Sensor Data Communication *H. Saputra, N. Vijaykrishnan, M. Kandemir, R.R. Brooks, and M.J. Irwin* *697*
Introduction • System Assumptions • Caching-Based Communication • Experimental Results • Spatial Locality • Related Work • Conclusions • Acknowledgments

36. Compiler-Directed Communication Energy Optimizations for Microsensor Networks *I. Kadayif, M. Kandemir, A. Choudhary, M. Karakoy, N. Vijaykrishnan, and M.J. Irwin* *711*
Introduction and Motivation • High-Level Architecture • Communication Optimizations • Experimental Setup • Results • Our Compiler Algorithm • Conclusions and Future Work

37. Sensor-Centric Routing in Wireless Sensor Networks *Rajgopal Kannan and S.S. Iyengar* *735*
Introduction • Sensor-Centric Reliable Routing • Reliable Routing Model • Results • Path Weakness • Simulation Results • Conclusions • Acknowledgments

© 2005 by Chapman & Hall/CRC

Sensor networks have gained great importance in a wide numbers of applications. A large variety of hostile environments require the deployment of large numbers of sensors for intelligent patient monitoring, object tracking, etc. Sensor nodes are typically battery operated and they possess a constraint on their energy, which is an important resource in sensor networks and other embedded systems. To maximize the sensor nodes' lifetime after their deployment, energy saving can be obtained by using static/dynamic power management techniques. In this chapter, we summarize the contributions by different authors on different aspects of power management.

Krishnan *et al.* emphasize on designing energy aware sensor systems. He introduces the major sources of energy consumption, the levels at which energy optimization can be performed, and present some representative optimizations that can be performed in the sensor network design space.

Swaminathan and Chakrabarty propose several node-level energy reduction techniques for the processor and I/O devices in real-time sensor nodes. The problem of scheduling device for minimum energy consumption is known to be NP-complete. They describe online algorithms to schedule the shutdowns and wakeup for I/O devices in sensor nodes by making a few simplifying assumptions to develop near optimal device schedules in reasonable periods of time.

Saptura *et al.* propose a new energy-efficient and secure communication protocol that exploits the locality of data transfer in distributed sensor networks. The focus of their work is on reducing the energy consumed due to the communication that happens between the nodes in a local cluster. Their approach also addresses another important issue of providing security for sensed data applications. Furthermore, they discuss other related works in energy consumption and improving security in sensor networks.

Kadayif *et al.* present a set of compiler-directed communication optimization techniques and evaluate from the energy perspective. More importantly, the authors focus on a wireless sensor network environment that processes array-intensive codes and has the following salient features: present energy behavior of a set of array-intensive applications on a wireless microsensor network; explain how source-level communication optimizations can reduce the energy consumption during communication; present a strategy where the communication time is overlapped with computation time; present a compiler algorithm that applies several communication optimizations in a unified framework for optimizing the energy spent in sensor network during execution.

Kannan and Iyengar structure the issue of reliable query routing (PQR) in a distributed sensor network. They describe a model for reliable data-centric routing with data-aggregation in sensor networks, where interest queries are disseminated through the network to assign sensing tasks to sensor nodes.

In summary, this section highlights issues regarding maximization of sensor node lifetime after deployment, by the use of energy saving power management techniques.

© 2005 by Chapman & Hall/CRC

33

Designing Energy-Aware Sensor Systems*

N. Vijaykrishnan, M.J. Irwin, M. Kandemir, L. Li, G. Chen, and B. Kang

33.1 Introduction

For various reasons, power and related metrics have become important design constraints in computing systems that range from low-end embedded systems to high-end servers. The limited improvements in the energy capacity of batteries and the rapid growth in the complexity of battery-operated systems have reduced the longevity of operation between battery recharges. Hence, optimizing energy consumption is of crucial importance in battery-driven mobile and embedded devices. Energy optimization is specifically important in sensor nodes that may not be accessible after deployment for replacing the batteries. Hence, energy optimization is critical in determining the longevity of the sensor network. Owing to constraints on physical size/weight, the energy capacity of the battery in sensor nodes is typically limited. Hence, energy optimization can either increase the lifetime of the node or enable more powerful operations at a sensor node. In order to tackle the limited battery capacities, many sensors scavenge energy from the external environment in addition to using batteries. However, the amount of power that can be harnessed this way is very limited (typically, in the range of 100 μW). Therefore, energy optimizations are important even in these environments. Energy consumption has a significant impact on the weight of the battery pack and consequently of the entire sensor node. For example, to limit the recharge interval to 10 h, it requires a 0.5 lb battery to operate a system that consumes 1 W.[1]

Power consumption also influences the design of tethered high-end systems that may be used as base stations to process the data from individual sensor nodes. While energy consumption is not a major issue in these systems that do not rely on batteries for operation, power consumption influences the amount of current drawn from the power supply and influences the power-supply grid design.

*This work is supported in part by the NSF Career Awards CCR-0093082 and CCR-0093085 and NSF Grant 0082064 and MARCO 98-DF-600 grant from GSRC. The authors also acknowledge the contributions of various graduate students from their lab who worked on several projects and whose results are abstracted here.

[1]This assumes the use of current Ni–Cd battery technology that offers a capacity of 20 W h/lb.

© 2005 by Chapman & Hall/CRC

It becomes more challenging and costly to design supply rails that supply larger currents in response to increased power consumption trends. Another concern arises due to the influence of power consumption on the thermal profile of the chip. A part of the power consumed from the supply rail is eventually dissipated in the form of heat. Thus, power consumption influences the cost of packaging and cooling deployed in a system. For example, it may be possible to employ a cheaper plastic packaging for low-power systems. Another consequence of the higher on-chip temperatures is the reduced reliability of the chip. Specifically, for every 10°C increase in junction temperature, the lifetime of a device is typically cut by half.

The objective of this chapter is to introduce the major sources of energy consumption, the levels at which energy optimization can be performed, and present some representative optimizations that can be performed in the sensor network design space. The rest of this chapter is organized as follows. Section 33.2 introduces the various sources of power consumption. Section 33.3 provides the different levels at which energy optimization can be performed in the design and operation of a system. Representative examples of energy reduction techniques are provided in Section 33.4. Finally, conclusions are provided in Section 33.5.

33.2 Sources of Power Consumption

Power is consumed in different portions of the sensor node, such as in the memory, communication blocks, mechanical parts, the compute processors, and the sensing elements. Since a significant portion of sensor nodes are composed of electronic circuits, we primarily focus on power consumption of CMOS VLSI circuits in this chapter.

Power consumption can be classified into the three major components: switching power P_{switch}, short-circuit power P_{sc} and leakage power P_{leakage}. The first two components are termed dynamic power, as power is consumed only when there is activity in the circuits, i.e. when signals change from zero to one and *vice versa*. The power consumed over the entire execution time is defined as the energy consumption. The power P consumption of CMOS circuits can be expressed as follows:

$$\begin{aligned} P &= P_{\text{switch}} + P_{\text{sc}} + P_{\text{leakage}} \\ &= C_{\text{L}} V_{\text{dd}}^2 \, p_{0\to1} \, f_{\text{clock}} + t_{\text{sc}} V_{\text{dd}} I_{\text{sc}} \, p_{0\to1} \, f_{\text{clock}} + k_{\text{design}} V_{\text{dd}} I_{\text{off}} N_{\text{transistor}} \end{aligned}$$

In this equation, C_{L} is the capacitive load of the circuit, V_{dd} is the supply voltage, $p_{0\to1}$ is the switching frequency, t_{sc} is the short-circuit time, f_{clock} is the clock frequency, and I_{sc} and I_{off} are the peak current during switching and the leakage current respectively. k_{design} is a design specific parameter and $N_{\text{transistor}}$ is the total number of transistors. Figure 33.1 illustrates the currents consumed due to the three components of power consumption in a simple CMOS circuit.

P_{switch} is consumed only when signals transition from 0 to 1, drawing current (I_{dyn}) from the power supply. However, switching power is dissipated as heat during both 0 to 1 and 1 to 0 transitions. Switching power can be reduced by decreasing the supply voltage, reducing the number of 0–1 transitions, reducing the capacitive load, or by decreasing the clock frequency. Switching power is the dominant source of energy consumption in current 130 nm technology designs.

P_{sc} is consumed when both the pull-down and pull-up stacks of the CMOS circuits are conducting (see I_{sc} in Figure 33.1) during the period when the inputs are transitioning from 0 to 1, or from 1 to 0. The key to reducing short-circuit power is slope engineering, which involves reducing the time spent in signal transitions from 0 to 1 and *vice versa*. Fast rising and falling edges can help reduce the short-circuit power. Further, reducing the number of transitions also helps to reduce short-circuit power. Short-circuit current forms a minor contribution in overall energy consumption in well-designed circuits.

Unlike dynamic power consumption, which has been the dominant form of energy consumption in CMOS-based circuits, leakage power P_{leakage} is consumed even in the absence of any switching activity

© 2005 by Chapman & Hall/CRC

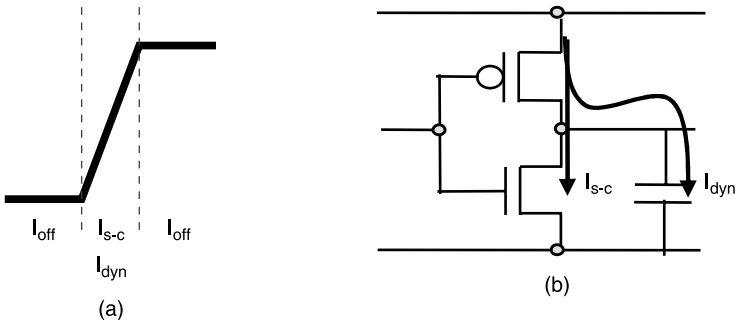


Figure 33.1. Components of power.

(see I_{off} in Figure 33.1). With advancing technology, leakage current has been increasing exponentially [1]. Leakage current per device is a function of various parameters, including temperature, supply voltage, threshold voltage, gate-oxide thickness, and doping profile of the device. Specifically, leakage current increases with higher supply voltages, higher temperature, lower threshold voltage, or thinner gate oxide. With continued progress in technology, devices are shrinking, resulting in thinner gate-oxides and lower threshold voltages. Further, the increasing complexity of chips has resulted in a larger number of transistors on the chip and also higher on-chip temperatures due to larger power dissipation. Consequently, managing leakage energy has become more important. Leakage power can be reduced by shutting off supply voltage to idle circuits or by increasing the threshold voltage of devices.

33.3 Power Optimizations: Different Stages of System Design

The power consumption due to the different sources can be reduced through optimizations spanning different stages of a system design. At the lowest level of system design level, the fabrication process used for designing the system can be improved to reduce power consumption. And at the highest level, the software application can be tuned to reduce energy consumption. A brief description of optimization possible at different stages of the system design is now given:

- *Process level.* At the process level, factors such as the choice of the material for gate oxide (high K versus low K dielectric), metal interconnect (e.g. copper versus aluminum), doping concentration, device dimensions, and device structure (single gate versus dual gate) influence the power consumption characteristics. This level of optimization mainly involves process engineers and device engineers.
- *Circuit/gate level.* This optimization level deals with circuit design techniques. Supply voltage and frequency-scaling techniques are the most widely used at this level. Voltage scaling yields considerable savings due to the quadratic dependence of power on supply voltage. However, the delay of the circuit increases with a reduction in the supply voltage due to a reduced driving strength (which is a function of the difference between supply and threshold voltage). Consequently, as supply voltage is reduced, the frequency of operation needs to be reduced. Since reducing supply voltage to the entire system can degrade performance, multiple-voltage techniques have been developed. In multiple-voltage designs, timing-critical modules are powered at high voltage level, while the other modules are powered using a low voltage. The leakage energy problem is solved similarly with multiple-threshold voltage circuits. Transistors in the critical path use low threshold voltages for high performance, whereas those in the noncritical path use high threshold voltages to reduce leakage.
- *Architectural level.* In this level, larger blocks, such as caches and functional units, are the main subject. In complex digital circuits, not all of the blocks perform meaningful operations every clock cycle. When a block is identified to be idle, it can be disabled to prevent useless but

© 2005 by Chapman & Hall/CRC

power-consuming transitions. Circuit techniques, such as clock gating, provide ways to apply this technique. To tackle leakage power, idle blocks can be turned off by power-gating.

- *Software level*. An operating system can achieve significant power reduction by performing energy-aware task scheduling and resource management. Processors these days adopt multiple power modes which are initiated by operating systems. These are collectively called dynamic power management. Another important system component is the compiler. Compilers traditionally have been studied to generate efficient codes in terms of performance. Many of the performance optimization techniques also reduce power consumption. For example, spill code reduction results in both performance improvement and power reduction. There have also been proposed power optimizing techniques which compromise performance. Power-aware instruction scheduling techniques can increase the total number of cycles. However, the performance degradation has to be limited.

Efforts at optimizing the energy consumption in sensor networks span these different levels and range from designing energy-efficient circuits for sensors to designing energy-efficient algorithms to map onto the sensor networks [2–4]. These energy optimizations can also be classified as those that are performed at system design time and those that are employed at run time. The optimizations performed at the process/circuit level are typically design-time solutions, whereas those at the architecture/software level can be either design-time or run-time optimizations.

33.4 Energy Reduction Techniques

Energy optimizations are typically targeted at the specific components of the sensor node that contribute a significant portion of the overall energy consumption. The main components of interest in a sensor node are the processor, the memory, the communication component, and the sensing elements. In the case of a mobile sensor, energy is also consumed by the components used to support locomotion.

In this section, we present representative energy optimizations targeted at the processor, memory, and communication components of the sensor node. Owing to the plethora of work on energy optimizations and due to space constraints, this section does not attempt to cover all efforts. Rather, our goal is to provide a sample of representative energy reduction techniques that are especially applicable to sensor systems.

33.4.1 Supply Voltage, Frequency and Threshold-Voltage Scaling

Many techniques for controlling both active and standby power consumption have been developed for processor cores [5]. Figure 33.2 shows the processor power design space for both active and standby power [6]. The second column lists techniques that are applied at design time, and thus are part of the

	Constant Throughput/Latency		**Variable Throughput/Latency**
	Design Time	**Sleep Mode**	**Run Time**
Active (Dynamic) [C V^2 f]	Logic design Trans sizing Multiple V_{DD}'s	Clock gating	DFS DVS DTM
Standby (Leakage) [V I_{off}]	Stack effect Multiple V_T's Multiple t_{ox}'s	Sleep trans Multi-V_{DD} Variable V_T Input control	Variable V_T

Figure 33.2. Processor power design space.

© 2005 by Chapman & Hall/CRC

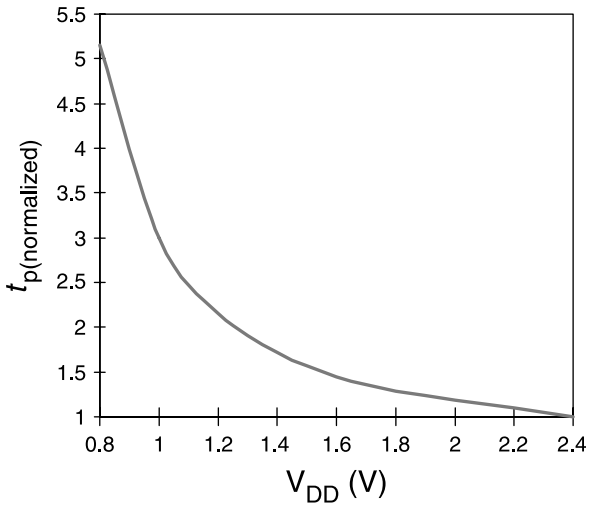


Figure 33.3. Delay as a function of V_{DD}.

circuit fabric. The last two columns list techniques that are applied at run time, the middle column for those cases when the component is idle and the last column when the component is in active use. (Run-time techniques can additionally be partitioned into those that reduce leakage current while retaining state and those that are state destroying.) The underlying circuit fabric must provide the "knobs" for controlling the run-time mechanisms — by either the hardware (e.g. in the case of clock gating) or the system software [e.g. in the case of dynamic voltage scaling (DVS)]. In the case of software control, the cost of transitioning from one state to another (both in times of energy and time) and the relative energy savings need to be provided to the software for decision making.

The best knob for controlling power is setting the supply voltage appropriately to meet the computational load requirements. While lowering the supply voltage has a quadratic impact on active energy, it decreases systems performance since it increases gate delay, as shown in Figure 33.3. Choosing the appropriate supply voltage at design time for the entire component will minimize (or even eliminate) the overhead of level converters that are needed whenever a module at a lower supply drives a module at a higher supply.

The most popular of the techniques for reducing active power consumption at run time is DVS, combined with dynamic frequency scaling (DFS). Most embedded and mobile processors contain this feature (triggered by thermal on-chip sensors when thermal limits are being approached or by the run-time system when the CPU load changes). Among the first approaches at exploiting DVS at run time was proposed by Weiser *et al.* [7]. Here, the authors propose a scheduling algorithm for minimizing energy consumption, where a process can be scheduled at different processor speeds, always keeping in mind the timely completion of such a task. Many tasks in sensor environments may need to meet some hard real-time constraints. For example, if there is a radiation sensor in a nuclear plant, then it needs to activate containment action with a given time constraint. In such environments, it is important to ensure that the reduction in performance due to voltage scaling does not affect the ability to meet hard real-time constraints. An example of a DVS approach meeting hard real-time systems is presented by Shin and Choi [8]. In their approach, the processor slack times are taken advantage of to power-down the processor, always paying attention to meet process deadlines.

DFS+DVS requires a power supply control loop containing a buck converter to adjust the supply voltage and a programmable phase-locked loop (PLL) to adjust the clock [6]. As long as the supply voltage is increased before increasing the clock rate or decreased after decreasing the clock rate, the system only need stall when the PLL is relocking on the new clock rate (estimated to be around 20 ms).

Several techniques have evolved recently for controlling subthreshold current, as shown in Figure 33.2. Since increasing the threshold voltage V_T decreases subthreshold leakage current

© 2005 by Chapman & Hall/CRC

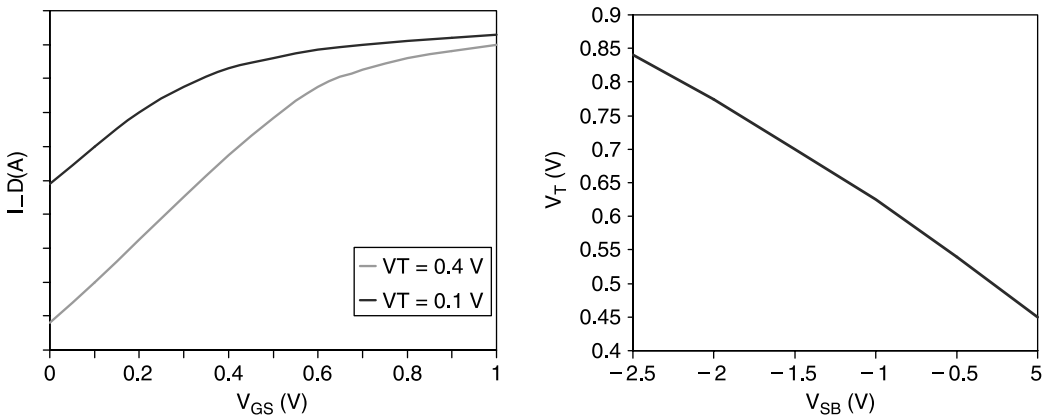


Figure 33.4. V_T effects. Note I_D is in exponential scale.

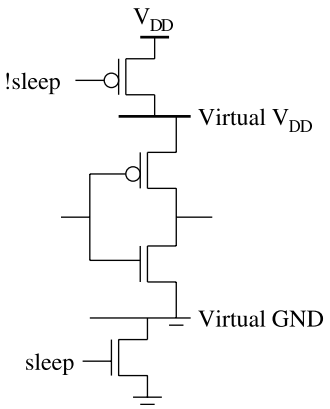


Figure 33.5. Gating supply rails.

(exponentially), adjusting V_T is one such technique. As shown in Figure 33.4, a 90 mV reduction in V_T increases leakage by an order of magnitude. Unfortunately, increasing V_T also negatively impacts gate delay. As with multiple supply voltages, multiple threshold voltages can be employed at design time or run time. At run time, multiple levels of V_T can be provided using adaptive body-biasing, where a negative bias on V_{SB} increases V_T, as shown in Figure 33.4 [9]. Simultaneous DVS, DFS, and variable V_T has been shown to be an effective way to trade off supply voltage and body-biasing to reduce total power — both active and standby — under variable processor loads [10].

Another technique that will apply to reducing standby power is the use of sleep transistors like those shown in Figure 33.5. Standby power can be greatly reduced by gating the supply rails for idle components. In normal mode (nonidle), the sleep transistors must present as small a resistance as possible (via sizing) so as not to affect performance negatively. In sleep mode (idle), the transistor stack effect [6] reduces leakage by orders of magnitude. Alternatively, standby power can be completely eliminated by switching off the supply to idle components. A sensor node employing such a technique will require system software that can determine the optimal scheduling of tasks on cores and can direct idle cores to switch off their supplies while taking into account the cost (both in terms of energy and time) of transitioning from the on-to-off and off-to-on states.

33.4.2 Shutting Down Idle Components

An effective means for reducing energy consumption is in shutting down the components. In the case of mechanical components such as the disk, the spinning of the disks is stopped to reduce power

© 2005 by Chapman & Hall/CRC

consumption. In the case of processor datapath or memory elements, shutting down the component could mean gating the supply voltage from the circuit to reduce leakage energy or gating the clocking circuit to reduce dynamic energy. Use of idleness to transition an entity to a low-energy mode is an issue that has been researched in the context of disks [11,12], network interfaces [13], and system events in general [14,15]. Many of these studies [11] have used past history (to predict future behavior) for effecting a transition. A similar strategy has recently been employed in the context of transitioning DRAM memory modules [16].

Shutting down idle components can be performed at different granularities. For example, supply gating can be performed for the entire memory, a single bank, or at an even finer granularity of a single memory block. The key issue is to determine when to shut down the component, balancing the energy requirements with other-tradeoffs, such as performance or reliability degradation. For example, frequent spinning up and down of the disks can increase the wear and tear on the disk head and impact on reliability. Reliability can also be a concern due to operation in the low-energy mode. For example, memory circuits operating at a low-leakage energy mode are more susceptible to bit flips induced by radiation effects. There can be performance implications as well. When disks are shut down, data reads can be delayed for the disk to spin up to normal speed.

33.4.2.1 Leakage Control Techniques for Memories

The memory element plays a vital role in sensor nodes as it captures the state of the system. However, maintaining the supply voltage to the entire memory will result in a significant drain on the energy resources of the sensor node due to leakage power. Hence, techniques that only maintain power supply to portions of the memory storing data will be important. Thus, we focus on various approaches that exist to reduce leakage energy in cache memories in order to motivate similar techniques that could be applied in sensor nodes.

Approaches that target reducing instruction cache leakage-energy consumption can be broadly categorized into three groups: (1) those that base their leakage management decisions on some form of performance feedback (e.g. cache miss rate) [17]; (2) those that manage cache leakage in an application insensitive manner (e.g. periodically turning off cache lines) [18–20]; and (3) those that use feedback from the program behavior [19,21,22].

The approach in category (1) is inherently coarse-grained in managing leakage, as it turns off large portions of the cache depending on a performance feedback that does not specifically capture cache line usage patterns. For example, the approach in (1) may indicate that 25% of the cache can be turned off because of very good hit rate, but it does not provide the guidance on which 75% of the cache lines are going to be used in the near future.

Approaches in category (2) turn off cache lines independent of the instruction access pattern. An example of such a scheme is the periodic cache line turn-off proposed by Flautner *et al.* [18], which turns off all cache lines after a fixed period. The success of this strategy depends on how well the selected period reflects the rate at which the instruction working set changes. Specifically, the optimum period may change not only across applications but also within the different phases of the application itself. In such cases, we either keep cache lines in the active state longer than necessary, or we can turn off cache lines that hold the current instruction working set, thereby impacting performance and wasting energy. Note that trying to address the first problem by decreasing the period will exacerbate the second problem. On the plus side, this approach is simple and has very little implementation overhead.

Another example of a fixed scheme in category (2) is the technique proposed by Kim *et al.* [20]. This technique adopts a bank-based strategy. In this approach, when an instruction fetches during execution moves from one cache memory bank to another, the hardware turns off the former and turns on the latter. Another technique in category (2) is the cache-decay-based approach [its adaptive variant falls in category (3)] proposed by Kaxiras *et al.* [19]. In this technique, a small counter is attached to each cache line which tracks its access frequency. If a cache line is not accessed for a certain number of cycles, then it is placed into the leakage saving mode. Altough this technique tries to capture the usage frequency of

© 2005 by Chapman & Hall/CRC

cache lines, it does not directly predict the cache line access pattern. Consequently, a cache line whose counter saturates is turned off even if it is going to be accessed in the next cycle. Since it is also a periodic approach, choosing a suitable decay interval is crucial if it is to be successful. In fact, the problems associated with selecting a good decay interval are similar to those associated with selecting a suitable turn-off period [18]. Consequently, this scheme can also keep a cache line in the active state until the next decay interval arrives even if the cache line is not going to be used in the near future. Finally, since each cache line is tracked individually, this scheme has more overhead.

The approaches in category (3) attempt to manage cache lines in an application-sensitive manner. The adaptive version of the cache-decay scheme [19] tailors the decay interval for the cache lines based on cache line access patterns. They start out with the smallest decay interval for each cache line to turn off cache lines aggressively and increase the decay interval when they learn that the cache lines were turned off prematurely. These schemes learn about premature turn-off by leaving the tags on at all times. Zhou *et al.* [21] also use tag information to adapt leakage management.

Zhang *et al.* [22] used an optimizing compiler to analyze the program to insert explicit cache line turn-off instructions. This scheme demands sophisticated program analysis and modification support, and needs modifications in the ISA to implement cache line turn-on/off instructions. In addition, this approach is only applicable when the source code of the application being optimized is available. In the Zhang *et al.* approach [22], instructions are inserted only at the end of loop constructs; hence, this technique does not work well if a lot of time is spent within the same loop. In these cases, periodic schemes may be able to transition portions of the loop that are already executed into a drowsy mode. Further, when only select portions of a loop are used, the entire loop is kept in an active state. Finally, inserting the turn-off instructions after a fast-executing loop placed inside an outer loop can cause performance and energy problems due to premature turn-offs.

Many of these leakage control approaches can be applied in the context of memory components in sensor nodes.

33.4.2.2 Multiple Low-Power Modes

Instead of just having two modes of turn-on/turn-off as in the case of the leakage control mechanism explained above, the system could support multiple low-power states. In this case, the task of choosing the appropriate power mode becomes more challenging. Typically, there are energy/performance trade-offs that need to be considered in selecting the mode of operation. In order to illustrate these trade-offs better and explain how multiple low-power modes are supported, we will use the energy/performance trade-offs that exist in operating the DRAM module in different power modes as an example.

Each energy mode is characterized by its *energy consumption* and the time that it takes to transition back to the active mode (*resynchronization time*). Typically, the lower the energy consumption the higher the resynchronization time [23]. These modes are characterized by varying degrees of the module components being active. The major components of a DRAM module are the clock generation circuitry, ROW (row address/control) decode circuitry and COL (column address/control) decode circuitry, control registers and power mode control circuitry, together with the DRAM core consisting of the precharge logic, memory cells, and sense amplifiers (see Figure 33.6). The clock-generation circuitry is used to generate two internal clock signals (TCK and RCK) that are synchronous with an external system clock (CLK) for transmitting read data and receiving write data/control signals. The packets received from the ROW and COL signals can also be used to switch the power mode of the DRAM. The details of the power modes are as follows:

- *Active.* In this mode, the DRAM module is ready for receiving the ROW and COL packets and can transition immediately to read or write mode. In order to receive these packets, both the ROW and COL demux receivers have to be active. As the memory unit is ready to service any read or write request, the resynchronization time for this mode is the least (zero units), and the energy consumption is the highest.

© 2005 by Chapman & Hall/CRC

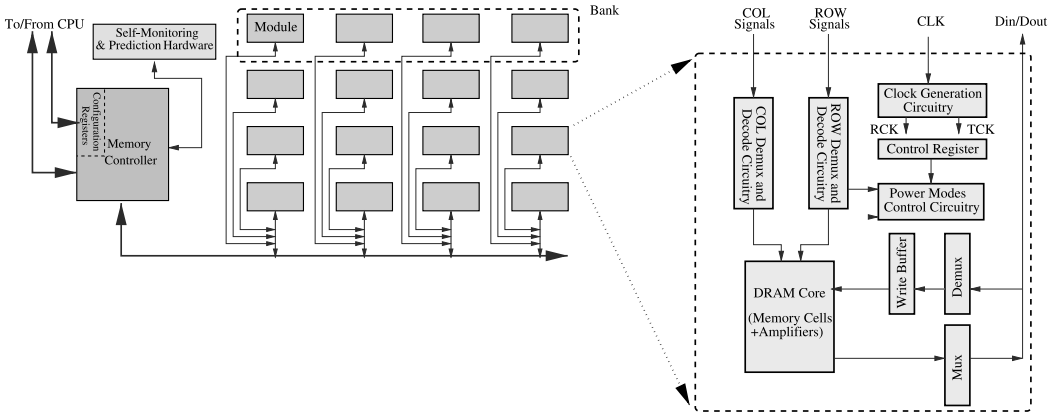


Figure 33.6. Memory system architecture.

- *Standby.* In this mode, the COL multiplexers are disabled, resulting in significant reduction in energy consumption compared with the active mode. The resynchronization time for this mode is typically one or two memory cycles. Some state-of-the-art RDRAM already exploit this mode by automatically transitioning into the standby mode at the end of a memory transaction [23].
- *Napping.* The ROW demux circuitry is turned off in this mode, leading to further energy savings over the standby mode. When napping, the DRAM module energy consumption is mainly due to the refresh circuitry and clock synchronization that is initiated periodically to synchronize the internal clock signals with the system clock. This mode can typically consume two orders of magnitude less energy than the active mode, with the resynchronization time being higher by an order of magnitude than the standby mode.
- *Power-down.* This mode shuts off the periodic clock synchronization circuitry, resulting in another order of magnitude saving in energy. The resynchronization time is also significantly higher (typically thousands of cycles).
- *Disabled.* If the content of a module is no longer needed, then it is possible to disable it completely (saving even refresh energy). There is no energy consumption in this mode, but the data are lost. One could envision transitioning out of disabled mode by reloading the data from an alternate location (perhaps another module or disk) and/or just performing write operations to such modules.

When a module in standby, napping, or power-down mode is requested to perform a memory transaction, it first goes to the active mode and then performs the requested transaction. Figure 33.7 shows possible transitions between modes (the dynamic energy consumed in a cycle is given for each node) in our model. The resynchronization times in cycles (based on a cycle time of 2.5 ns) are shown along the arrows.

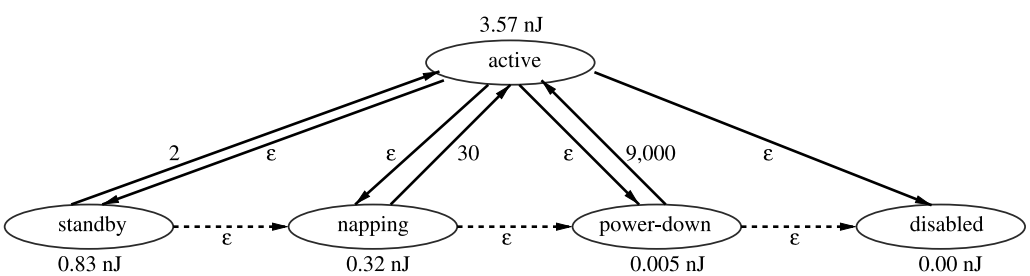


Figure 33.7. Power modes utilized.

© 2005 by Chapman & Hall/CRC

It is evident from the operation of the DRAM power modes that there are significant performance/power trade-offs in selecting the power mode to operate. Schemes that have been employed to utilize these power modes typically predict the duration of the idleness of the memory module and then select the power mode such that the overheads in terms of energy/time in transitioning to and from a low-power mode amortize the benefit gained in operating in the low-power mode. Examples of such the approaches are the constant-threshold predictor (CTP) and the history-based threshold predictor (HBP) proposed by Delaluz *et al.* [16].

The rationale behind the CTP is that if a memory module has not been accessed in a while, then it is not likely to be needed in the near future (i.e. inter-access times are predicted to be long). A threshold is used to determine the idleness of a module, after which it is transitioned to a lower energy mode. Hence, using this scheme, a memory module is progressively transitioned from the active mode to the power-down mode. The thresholds are typically chosen to be larger if the penalties for recovering to the active mode are large. There are two main problems with this CTP approach. First, we gradually decay from one mode to another (i.e. to get to power-down, we go through standby and napping), though one could have directly transitioned to the final mode if we had a good estimate. Second, we pay the cost of resynchronizing on a memory access if the module has been transitioned. In the HBP, we estimate the inter-access time, directly transition to the best energy mode, and activate (resynchronize) the module so that it becomes ready by the time of the next estimated access.

Since sensor nodes are expected to have many components that will support multiple low-power modes, we anticipate techniques such as CTP and HBP to be pertinent in this environment. An important consideration in devising such schemes is to estimate the effect of using these modes not on just individual components, but also on the entire sensor node. For example, performance degradations that result in order to save energy in the memory module should not be offset by the increased energy consumption in the rest of the components due to the prolonged operation.

33.4.2.3 Adaptive Communication Hardware

Similar to the memory elements, other components can also be partially shutdown to conserve power consumption. For example, portions of the pipeline in the processor can be shut down based on the data width of the computation, or the number of bits used by the analog-to-digital converter (ADC) can be varied based on the desired accuracy of the data. We show how an ADC, one of the components used by the communication system, can be adapted to the required channel conditions.

The architecture of a traditional pipelined ADC can be easily modified for adapting its structure to the required resolution to conserve energy consumption. Each stage has the same function and its input is only correlated to the previous stage's output. The most significant bit is defined by the leftmost stage of the ADC. Similarly, the least significant bit is defined by the rightmost stage of the ADC. If unused stages could be shut down, then a significant power saving can be expected. Supply gating is used as the mechanism for reducing power consumption in the unused segments. There is no leakage current or static power consumption in the supply gated units. However, a recovery latency of 350 ns is incurred for activating a shutdown segment. Based on the required resolution, supply gating is applied to selected units in our adaptive system shown in Figure 33.8. Further, the input gates of the unused stages are disabled.

In order to determine the desired number of bits that need to be used by the ADC, the channel condition is determined using the receiving power. Next, the symbol rate is determined based on the channel condition to ensure the required bit error rate. This, in turn, can be used to determine the number of stages of the adaptive pipelined ADC that can be shut down while still meeting the required resolution. Higher symbol rates are possible only when channel conditions are good and when the ADC supports higher resolution to detect the symbol correctly. Thus, when symbol rates are lower, it is possible to conserve power consumption by shutting down portions of the ADC.

Through the different examples that were presented in this section, we have reiterated the importance of shutting down idle components as an important scheme in conserving energy consumption.

© 2005 by Chapman & Hall/CRC

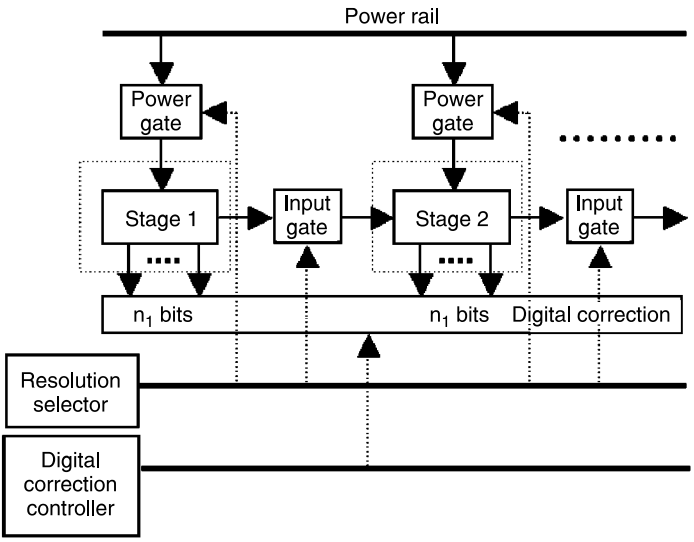


Figure 33.8. Adaptive bitwidth ADC.

33.4.3 Computational Offloading

In wireless sensor nodes, it is important to optimize both computation and communication energy. There are different opportunities in such a system that allow trade-offs between computation and communication energy costs. A lot of research has been carried out to exploit such tradeoffs to reduce the overall energy consumption, e.g. [24–26]. For example, Flinn and Satyanarayanan [24] have proposed an environment where applications dynamically modify their behaviors to conserve energy by lowering data fidelity and adjusting the partition of computation tasks between client and server.

A major goal of these approaches is to offload complex computational tasks from an energy-constrained node to a more powerful server. However, the cost of offloading the task to a remote server involves communication costs for transmitting the code and data, as well as for receiving the results of the computation. Hence, the challenge in these schemes is deciding on whether the energy reduced by decreasing the number of computations is greater than the overhead incurred for communication. Based on these tradeoffs, a decision should be made as to where the computation will be performed, i.e. locally or remotely. While it is possible to make some of these decisions statically, the energy cost of computation and communication can vary based on the operating conditions and user-supplied input, for which accurate information can only be obtained at run time. For example, the power amplifier of the transmitter in the mobile client should be tuned according to its distance from the server and the channel interference. This setting affects the energy cost of transmitting a bit. Similarly, the user input can also influence the energy costs of local computation and offloading. As an example, the complexity of a computation can depend on the magnitude of an input parameter (e.g. image size in an image-processing application).

The energy consumption due to communication is dominated by either the receiving or transmitting energy based on the type of sensor network employed. When communicating over small distances (less than 5 m) in micro-sensor networks, the cost of transmission is relatively small. In contrast, macro-sensor networks need to transmit larger distances, where transmission energy is significantly larger than receiving energy. Hence, when considering computational offloading algorithms in micro-networks, it is important to reduce the idle energy expended by the receivers when they are idling snooping for data. The need to offload computation will require the receivers to be active in order to receive the results back and may in itself cause a significant portion of overall energy. Hence, proposed approaches for

© 2005 by Chapman & Hall/CRC

computational offloading estimate the amount of time that the server will require for completing the offloaded computation before it returns the results and shuts down the receiver for that period.

33.4.4 Energy-Aware Routing

Energy consumed due to communication between the different sensor nodes can consume a significant portion of the energy. Since, the energy consumption for transmitting is a function of the distance between the communicating nodes, there have been several efforts focusing on devising routing algorithms that minimize the distance of communication. A common approach to reducing energy consumption is the use of multi-hop routing. In multi-hop routing, nodes between the source and sink nodes of the communication are used as intermediary hops in the transmission. This approach is beneficial, as the energy increases faster than linear with distance of transmission. Hence, having multiple transmissions of smaller distances is more energy efficient than a single transmission of a longer distance.

Another approach to reducing the energy consumption is to partition sensors network into several clusters [2] and limit the communication outside the cluster to a selected node or a set of nodes in the cluster. This approach confines the communication of the other nodes to a small distance, and incorporates some intelligent processing within the local cluster to reduce the amount of communication outside the cluster.

There are different criteria for selecting the routing algorithms used in a sensor network. For example, a minimum energy path to reach the destination may not always be desirable, as the optimal path to route a packet may require the use of an intermediary node that is running low on battery. In such a case, a higher energy path that avoids this node may be more desirable in order to prevent this node from becoming dead. Shah and Rabaey [27] show that a sub-optimal path chosen based on a probabilistic approach, instead of always using the optimal path, is preferable to prevent nodes from dying.

33.5 Conclusions

This chapter has emphasized the need for energy optimization and emphasizes the need for optimizations to span different design stages of the system. Optimizing the system energy requires a concerted effort spanning from the choice of the underlying process technology for fabrication of the system to the run-time software deployed on the sensor node. We have also introduced the major sources of energy consumption and explained various approaches for reducing the energy consumption due to these parts. Devising new energy optimizations continues to remain an important issue for sensor networks. Techniques aimed at limiting the energy consumption hold the key for increased deployment of sensor networks.

References

[1] Butts, J.A. and Sohi, G., A static power model for architects, in *Proceedings of the 33rd Annual International Symposium on Microarchitecture*, December 2000, 191.

[2] Estrin, D., Wireless sensor networks: application driver for low power distributed systems, in *Proceedings of ISLPED,* August 2001, 194.

[3] Heinzelman, W. *et al.*, Energy-scalable algorithms and protocols for wireless microsensor networks, in *Proceedings of International Conference on Acoustics, Speech, and Signal Processing,* June 2000, 3722.

[4] Chandrakasan, A. *et al.*, Design considerations for distributed microsensor systems, in *Proceedings of IEEE 1999 Custom Integrated Circuits Conference*, May 1999, 279.

[5] Brodersen, R. *et al.*, Methods for true power minimization, in *International Conference on Computer Aided Design*, November 2002, 35.

[6] Rabaey, J., *Digital Integrated Circuits: A Design Perspective*, Prentice Hall, 2003.

© 2005 by Chapman & Hall/CRC

[7] Weiser, M. *et al.*, Scheduling for reduced CPU energy, in *Proceedings of the 1st Symposium on Operating Systems Design and Implementation*, November 1994, 13.
[8] Shin, Y. and Choi, K., Power conscious fixed priority scheduling for hard real-time systems, in *Proceedings of the 36th Design Automation Conference* (DAC'99), 1999, 134.
[9] Duarte, D. *et al.*, Evaluating run-time techniques for leakage power reduction techniques, in *Asia–Pacific Design Automation Conference*, January 2001, 2, 31.
[10] Martin, S. *et al.*, Combined dynamic voltage scaling under adaptive body biasing for lower power microprocessors under dynamic load, in *International Conference on Computer Aided Design*, November 2002, 712.
[11] Li, K. *et al.*, A quantitative analysis of disk drive power management in portable computers, in *Proceedings of Winter Usenix*, 1994.
[12] Douglas, F. *et al.*, Thwarting the power-hungry disk, in *Proceedings of Winter Usenix*, 1994.
[13] Stemm, M. and Katz, R.H., Measuring and reducing energy consumption of network interfaces in hand-held devices, *IEICE Transactions on Communications*, Special Issue on Mobile Computing, 80(8), 1125, 1997.
[14] Benini, L. *et al.*, Monitoring system activity for OS-directed dynamic power management, in *Proceedings of ACM ISLPED'98*, Monterey, CA, 1998, 185.
[15] Benini, L. *et al.*, System-level power estimation and optimization, in *Proceedings of ACM ISLPED'98*, Monterey, CA, 1998, 173.
[16] Delaluz, V. *et al.*, DRAM energy management using hardware and software directed power mode control, in *Proceedings of the International Conference on High Performance Computer Architecture (HPCA)*, January 2001, 159.
[17] Powell, M.D. *et al.*, Reducing leakage in a high-performance deep-submicron instruction cache, *IEEE Transactions on VLSI*, 9(1), 77, 2001.
[18] Flautner, K. *et al.*, Drowsy caches: simple techniques for reducing leakage power, in *Proceedings of the 29th International Symposium on Computer Architecture*, Anchorage, AK, May 2002, 148.
[19] Kaxiras, S. *et al.*, Cache decay: exploiting generational behavior to reduce cache leakage power, in *Proceedings of the 28th International Symposium on Computer Architecture*, Sweden, June 2001, 240.
[20] Kim, N. *et al.*, Drowsy instruction caches: leakage power reduction using dynamic voltage scaling and cache sub-bank prediction, in *Proceedings of the 35th Annual International Symposium on Microarchitecture*, November 2002, 219.
[21] Zhou, H. *et al.*, Adaptive mode control: a static power-efficient cache design, in *Proceedings of the 2001 International Conference on Parallel Architectures and Compilation Techniques*, September 2001, 61.
[22] Zhang, W. *et al.*, Compiler-directed instruction cache leakage optimization, in *Proceedings of the 35th Annual International Symposium on Microarchitecture*, November 2002, 208.
[23] 128/144-MBit Direct RDRAM Data Sheet, Rambus Inc., May 1999.
[24] Flinn, J. and Satyanarayanan, M., Energy-aware adaptation for mobile applications, in *The 17th ACM Symposium on Operating Systems Principles*, Kiawah Island Resort, SC, December 1999, 48.
[25] Li, Z. *et al.*, Computation offloading to save energy on handheld devices: a partition scheme, in *International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, November 2001, 238.
[26] Kremer, U. *et al.*, A compilation framework for power and energy management on mobile computers, in *The 14th International Workshop on Parallel Computing*, August 2001.
[27] Shah, R.C. and Rabaey, J., Energy aware routing for low energy *ad hoc* sensor networks, in *IEEE Wireless Communications and Networking Conference (WCNC)*, Orlando, FL, March 17–21, 2002, 350.
[28] Sinha, A. and Chandrakasan, A., Operating system and algorithmic techniques for energy scalable wireless sensor networks, in *Proceedings of the 2nd International Conference on Mobile Data Management,* January 2001.

© 2005 by Chapman & Hall/CRC

[29] Ye, W. *et al.*, An energy-efficient MAC protocol for wireless sensor networks, in *Proceedings of the 21st International Annual Joint Conference of the IEEE Computer and Communications Societies,* New York, NY, USA, June 2002, 1567.

[30] Kuroda, T., Optimization and control of v_{DD} and v_t for low power, high speed CMOS design, in *International Conference on Computer Aided Design*, November 2002, 28.

© 2005 by Chapman & Hall/CRC

34

Operating System Power Management

Vishnu Swaminathan and Krishnendu Chakrabarty

34.1 Introduction

Energy consumption is an important design consideration for wireless sensor networks. These networks are useful for a number of applications, such as environment monitoring, surveillance, and target detection and localization. The sensor nodes in such applications operate under limited battery power. Sensor nodes also tend to be situated at remote and/or inaccessible locations, and hence the cost of replacing battery packs is high when the batteries that power them fail.

One approach to reduce energy consumption is to employ low-power hardware design techniques [1–3]. These design approaches are static, in that they can only be used during system design and synthesis. Hence, these optimization techniques do not fully exploit the potential for node-level power reduction under changing workload conditions, and their ability to trade off performance with power reduction is thus inherently limited. An alternative and more effective approach to reducing energy in embedded systems and sensor networks is based on dynamic power management (DPM), in which the operating system (OS) is responsible for managing the power consumption of the system.

Many wireless sensor networks are also designed for *real-time* use. Real-time performance is defined in terms of the ability of the system to provide real-time temporal guarantees to application tasks that request such guarantees. These systems must, therefore, be designed to meet both functional and timing requirements [4]. Energy minimization adds a new dimension to these design criteria. Thus, while energy minimization for sensor networks is of great importance, energy reduction must be carefully balanced against the need for real-time responsiveness.

Recent studies have shown that the CPU and the I/O subsystem are major consumers of power in an embedded system; in some cases, hard disks and network transceivers consume as much as 20% of total system power in portable devices [5,6]. Consequently, CPU-centric and I/O-centric DPM techniques have emerged at the forefront of DPM research for wireless sensor networks.

© 2005 by Chapman & Hall/CRC

34.1.1 CPU-Centric DPM

Designers of embedded processors that are used in sensor nodes now include variable-voltage power supplies in their processor designs, i.e. the supply voltages of these processors can be adjusted dynamically to trade off performance with power consumption. Dynamic voltage scaling (DVS) refers to the method by which quadratic savings in energy are obtained through the run-time variation of the supply voltage to the processor.

It is well known that the power consumption of a CMOS circuit exhibits a cubic dependence on the supply voltage V_{dd}. However, the execution time of an application task is proportional to the sum of the gate delays on the critical path in a CMOS processor. Since gate delay is inversely proportional to V_{dd}, the execution time of a task increases with decreasing supply voltage. The energy consumption of the CMOS circuit, which is the product of the power and the delay, therefore exhibits a quadratic dependence on V_{dd}. In embedded sensor nodes, where peak processor performance is not always necessary, a drop in the operating speed (due to a reduction in operating voltage) can be tolerated in order to obtain quadratic reductions in energy consumption. This forms the basis for DVS; the quadratic dependence of energy on V_{dd} has made it one of the most commonly used power reduction techniques in sensor nodes and other embedded systems. When processor workload is low, the OS can reduce the supply voltage to the processor (with a tolerable drop in performance) and utilize the quadratic dependence of power on voltage to reduce energy consumption.

34.1.2 I/O-Centric DPM

Many peripheral devices possess multiple power states — usually one high-power working state and at least one low-power sleep state. Hardware-based timeout schemes for power reduction in such I/O devices have been incorporated into several device designs. These techniques shut down devices when they have been idle for a prespecified period of time. A device that has been placed in the sleep state is powered up when a new request is generated.

With the introduction of the ACPI standard in 1997, the operating system was provided with the ability to switch device power states dynamically during run time, thus leading to the development of several new types of DPM technique. Predictive schemes use various system parameters to estimate the lengths of idle periods for devices. Stochastic models with different probabilistic distributions have been used to estimate the times at which devices can be switched between power states. The goals of these methods, however, are to minimize the response times of devices. Indeed, many such probabilistic schemes see widespread use in portable and interactive systems, such as laptop computers. However, their applicability in sensor systems, many of which require real-time guarantees, is limited due to a drawback inherent to probabilistic methods.

Switching between device power states incurs a time penalty, i.e. a device takes a certain amount of time to transition between its power states. In hard real-time systems, where tasks have firm deadlines, device switching must be performed with caution to avoid the potentially disastrous consequences of missed deadlines. The uncertainty that is inherent in probabilistic estimation methods precludes their use as effective device-switching algorithms in hard real-time systems whose behavior must be predicatable with a high degree of confidence. Current-day practice consists of keeping devices in real-time systems powered up during the entirety of system operation; the critical nature of I/O devices operating in real time prohibits the shutting down of devices during run time.

In this chapter, we describe several node-level energy reduction methods for wireless sensor networks. The first algorithm targets the processor in a sensor node. This algorithm is implemented on a laptop equipped with an AMD Athlon 4 processor and running the Real-time Linux (RT-Linux) operating system. Experimental power measurements validate and support our simulation results. A significant amount of energy is saved using the algorithm described here. We describe an optimal offline algorithm that generates device schedules for minimum energy consumption of I/O devices in hard real-time sensor nodes. The problem of scheduling devices for minimum energy consumption is

© 2005 by Chapman & Hall/CRC

known to be $\mathcal{NP}$-complete. However, by making a few simplifying assumptions, online algorithms can be developed to generate near-optimal device schedules in reasonable periods of time. Here, we describe two such online algorithms to schedule the shutdowns and wake-ups for I/O devices in sensor nodes that require hard real-time temporal guarantees.

34.2 Node-Level Processor-Oriented Energy Management

We are given a set $R = \{r_1, r_2, \ldots, r_n\}$ of n tasks. Associated with each task $r_i \in R$ are the following parameters: (i) an arrival time a_i, (ii) a deadline d_i, and (iii) a length l_i (represented as the number of instruction cycles). Each task is placed in the ready queue at time a_i and must complete its execution by its deadline d_i. The tasks are not preemptable. The CPU can operate at one of k voltages: $V_1, V_2, \ldots, V_k$. Depending on the voltage level, the CPU speed may take on k values: $s_1, s_2, \ldots, s_k$. The supply voltage to the CPU is controlled by the OS, which can dynamically switch the voltage during run time. The energy E_i consumed by task r_i is proportional to $v_i^2 l_i$. The problem we address is defined as follows:

> $\mathcal{P}_{\text{cpu}}$: given a set R of n tasks, and for each task $r_i \in R$, (i) a release time a_i, (ii) a deadline d_i, and (iii) a length l_i, and a processor capable of operating at k different voltages $V_1, V_2, \ldots, V_k$ with corresponding speeds $S_1, S_2, \ldots, S_k$, determine a sequence of voltages $v_1, v_2, \ldots, v_n$ and corresponding speeds $s_1, s_2, \ldots, s_n$ for the task set R such that the total energy consumed $\sum_{i=1}^{n} v_i^2 l_i$ by the task set is minimized, while also attempting to meet as many task deadlines as possible.

34.2.1 The LEDF Algorithm

LEDF is an extension of the well-known earliest deadline first (EDF) algorithm [7]. The algorithm maintains a list of all released tasks called the ready list. These tasks have an absolute deadline associated with them that is recalculated at each release based on the absolute time of release and the relative deadline. When tasks are released, the task with the earliest deadline is selected for execution. A check is performed to see if the task deadline can be met by executing it at a lower voltage (speed). Each speed at which the processor can run is considered in order from the lowest to the highest. For a given speed, the worst-case execution time of the task is calculated based on the maximum instruction count. If this execution time is too high to meet the current absolute deadline for the task, then the next higher speed is considered. Otherwise, a schedulability test is applied to verify that all ready tasks will be able to meet their deadlines when the current earliest-deadline task is run at a lower speed. The test consists of iterating down the ordered list of tasks and comparing the worst-case completion time for each task (at the highest speed) against its absolute deadline. If any task will miss its deadline, then the selected speed is insufficient and the next higher speed for the current task is considered. If the deadlines of all tasks in the ready list can be met at the highest speed, then LEDF assigns the lower voltage to the task and the task begins execution. When the task completes execution, LEDF again selects the task with the nearest deadline to be executed. As long as there are tasks waiting to be executed, LEDF schedules the one with the earliest absolute deadline for execution. Figure 34.1 describes the algorithm in pseudocode form.

For a processor with two speeds, the LEDF algorithm has a computational complexity of $O(n \log n)$, where n is the total number of tasks. The worst-case scenario occurs when all n tasks are released at time $t = 0$. This involves sorting n tasks in the ready list and then selecting the task with the earliest deadline for execution. When more than two speeds are allowed, the complexity of LEDF becomes $O(n \log n + kn)$, where k is the number of speed settings that are allowed.

34.2.2 Implementation Testbed

34.2.2.1 Hardware Platform

The power measurement experiments were conducted on a laptop with an AMD Mobile Athlon 4 processor. AMD's PowerNow! technology offers greater flexibility in setting both frequencies and core voltages [8]. The 1.1 GHz Mobile Athlon 4 processor can be set at several core voltage levels ranging

© 2005 by Chapman & Hall/CRC

Procedure LEDF()

t_c: current time;
$S_h > S_{l1} > S_{l2} > \ldots S_{lm}$: Available processor speeds
`schedulable = 1`
1. **if** ready_list $\neq$ NULL
2. Sort task deadlines in ascending order;
3. Select task τ_i with earliest deadline;
4. **for** $S = S_{lm}$ to S_h
5. **if** $t_c + \frac{l_i}{S} \leq d_i$ **then**
6. $t = t_c + \frac{l_i}{S}$
7. **for** each task τ_u that has not completed execution
8. **if** $t + \frac{l_u}{S_h} \leq d_u$ **then**
9. $t = t + \frac{l_u}{S_h}$
10. **else**
11. `schedulable = 0`
12. **break**
13. **endfor**
14. **if** `schedulable` = 1 **then**
15. Schedule τ_i at S
16. **break**
17. **endif**
18. **endfor**

Figure 34.1. The LEDF algorithm.

Table 34.1. Speed and voltage settings for the Athlon 4 processor

Power state	Speed (MHz)	Voltage (V)
1	1100	1.4
2	900	1.35
3	700	1.25

from 1.2 to 1.4 V in 0.05 V increments. For each core voltage there is a predetermined maximum clock frequency. The power states we chose to use in our scheduler and simulations are shown in Table 34.1. Although we use only three speeds in our experiments, an extension to using all five available speeds appears to be quite straightforward.

PowerNow! technology was developed primarily to extend battery life on mobile systems. We therefore conducted our experiments on a laptop system rather than a desktop PC. Instead of inserting a current probe into the laptop, we opted to simply measure system power during the experiments. The laptop's system power is drawn from the power converter at approximately 18.5 V DC. Instead of using an oscilloscope or digital ammeter to take exact CPU power measurements at very high frequencies, we chose the simpler approach of using a large capacitor to average out the DC drawn by the entire laptop. This method works primarily due to the periodic nature of our tests. In a periodic real-time system, the power drawn over one hyperperiod is roughly the same as the power drawn over the next hyperperiod as long as no tasks are added or removed from the task set. Since a fairly large amount of energy needs to be sourced and sunk by the capacitor at the different processor speeds and activity levels, we used a 30 V DC 360 mF capacitance (a 160 mF and a 200 mF capacitor in parallel). This capacitance proved capable of averaging current loads for power state periods ranging up to hundreds of milliseconds. When the processor power state switches at a lower rate than this, the current measurements taken between the AC/DC converter and the voltmeter readings fluctuate. Figure 34.2 illustrates our experimental hardware setup.

34.2.2.2 Software Architecture

We used RT-Linux [9] as the OS for our experiments. In addition to providing real-time guarantees for tasks and a periodic scheduling system, RT-Linux also provides a well-documented method of changing

© 2005 by Chapman & Hall/CRC

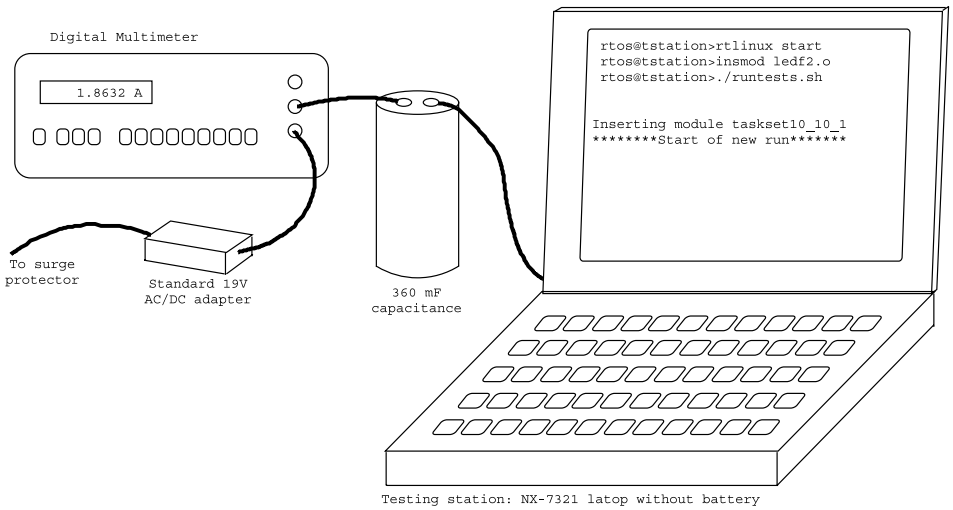


Figure 34.2. Illustration of the experimental setup.

the scheduling policies. An elegant modular interface allows for easy adaptation of the scheduler module to use LEDF and then load and unload it as necessary. We used this feature of RT-Linux to swap LEDF for a regular EDF scheduler during power comparisons. Furthermore, RT-Linux uses Linux as its idle task, providing a very convenient method of control and evaluation for the execution of the real-time tasks.

LEDF sorts all tasks by their absolute deadlines and chooses the task with the earliest deadline first. If there are no real-time tasks pending, the Linux/idle task is chosen and run at the lowest available speed. A timeout is then set to preempt the idle task at the next known release time. Once a speed is identified for a task, the switching code is invoked if the processor is not already operating at that speed.

Switching the power state of a Mobile Athlon 4 processor simply consists of writing to a model-specific register (MSR). The core voltage and clock frequency at which the processor is to be set are encoded into a 32-bit word along with three control bits. Another 32-bit word contains the stop–grant timeout count (SGTC), which represents the number of 100 MHz system clocks during which the processor is stalled for the voltage and frequency changes. The maximum phase-locked loop (PLL) synchronization time is 50 μs and the maximum time for ramping the core voltage appears to be 100 μs. Calling the WRMSR macro then instruments the power state change. For debugging, the RDMSR macro was used with a status MSR to retrieve the processor's power state. Decoding the two 32-bit word values reveals the maximum, current, and default frequency and core voltage.

The RT-Linux high-resolution timer used for scheduling is based (in x86 systems) on the time-stamp counter (TSC). The TSC is a special counter introduced by Intel that simply counts clock periods in the CPU since it was started (boot-time). The *gethrtime*() RT-Linux method (and all methods derived from it) convert the TSC value into a time value using the recorded clock frequency. Thus, a simple calculation to determine time in nanoseconds from the TSC value would be the product of TSC and clock period. Since RT-Linux was initially developed without the need for dynamic frequency switching, the speed used for the calculation of time is set at boot time and never changed. Thus, when the processor is slowed to a low-power state with a lower clock frequency, the TSC counts at a lower rate. However, the *gethrtime*() method is oblivious to this and the measurement of time slows down proportionally. It is not clear what happens to the TSC, and thus how to measure time, during a speed switch. The TSC does appear to be incremented during some part of the speed switch, but the count is not a reliable means of measuring time. Recalibrating the rate at which the TSC is incremented appears to be a nontrivial task which requires extensive rewriting of the RT-Linux timing code. Therefore, we chose to track time from within the LEDF module.

© 2005 by Chapman & Hall/CRC

34.2.3 Experimental Results

We now present data from our power measurement experiments. In these experiments, we measure the total system power consumption of the laptop. Knowledge of CPU power savings, however, is useful in generalizing the results. CPU power savings can easily be derived from a set of experiments. In order to isolate the power used by the processor and system board, we can turn off all system components except the CPU and system board. We can then take a power reading when the CPU is halted. This power measurement represents the total system power excluding CPU power. We can then subtract this base power from all future power readings in order to obtain CPU power alone. However, halting a processor is far more complex than simply issuing a "HLT" instruction. Decoupling the clock from the CPU involves handshaking between the CPU and the Northbridge. We were unable to obtain sufficient documentation to implement this.

As an alternative method of estimating power drawn by the system board and components, the power consumption of the CPU with maximum load can be calculated from system measurements at two power states. This can be done by devising tests to isolate power drawn by the LCD screen, hard drive, and the portion of the system beyond our control. Once an estimate for system power is available, we can eliminate this from all our readings to get an approximation of the fraction of CPU power being saved.

Ratios for power consumption in different states can be calculated using the well-known relationship for CMOS power consumption, i.e. $P = faCV_{\text{dd}}^2$, where P is the power, f is the frequency of operation, a is the average switching activity, C is the switching capacitance, and V_{dd} is the operating voltage. The switching capacitance and average switching activity is constant for the same processor and software, so we only consider the frequency and the square of the core voltage. It is also reasonable to assume that other components of the laptop (the screen and hard disk, for example) draw approximately the same current regardless of the CPU operating voltage. We therefore calculate that power state 2 uses approximately 76% as much power as power state 1, and power state 3 uses only 50.7% as much power as the maximum power state. The minimum power configuration for this processor is 300 MHz at 1.2 V, which consumes only 20% of the power consumed in the maximum power state.

In our case, we chose to compare a fully loaded processor operating at 700 MHz (with a core voltage of 1.25 V) and at 1100 MHz (with a core voltage of 1.4 V). The 700 MHz configuration uses $(700 \times 1.25^2)/(1100 \times 1.4^2)$, or 50.73% as much CPU power as the 1100 MHz configuration. For a given task running at 1100 MHz, the observed current consumption was 2.373 A. For the same task running at 700 MHz, we observed a current reading of 1.647 A. Assuming that the current consumption of the other components was approximately the same during both runs, the difference in CPU current consumption is 0.726 A. This means:

$$
\begin{aligned}
I_{1100} - I_{700} &= 0.726 \\
\Rightarrow I_{1100} - 0.5073 I_{1100} &= 0.726 \\
\Rightarrow I_{1100} &= 1.474\,\text{A}
\end{aligned}
$$

In other words, a measured difference of $(2.373 - 1.647) = 0.726$ A of current implies that the fully loaded CPU operating at 1100 MHz draws approximately 1.474 A. Knowing this, we deduce from the information in Table 34.2 that the system board and basic components draw approximately 0.456 A, and that under normal operation the system (including the disk drive and display) draws about 0.976 A in addition to the load from the CPU. This estimation, although approximate, provides a useful method of isolating energy used by the CPU for various utilizations and scheduling algorithms.

We performed several experiments with three different versions of the scheduling algorithm and different task sets at various CPU utilization levels. We constructed a pseudorandom task generator to generate our test sets. Using the task generator, we created several random sets of tasks. The release times of the tasks are set to the beginning of a period and deadlines to the end of a period. Computation requirements for the tasks are chosen randomly and then scaled to meet the target utilization.

© 2005 by Chapman & Hall/CRC

Table 34.2. Current consumptions of various system components

CPU (1100 MHz)	Screen	Disk	Current drawn (A)
Idle	Off	STBY	1.5
Idle	Off	On	1.54
Idle	On	STBY	1.91
Idle	On	Sleep	1.9
Idle	On	On	1.97
Max Load	Off	STBY	1.93
Max Load	On	On	2.45

The tests programs consist of multiple threads that execute ''for'' loops for specified periods of time. The time for which these threads run can be determined by examining the assembly-level code for each iteration of a loop. Each loop consists of five assembly language instructions which take one cycle each to execute. The random task set generator takes this into account when generating the task sets.

The simulator is a simple PERL program that reads in task data and generates the schedule which would be generated by the LEDF scheduler. It then takes user-supplied baseline power measurements and uses them to compute the power consumption of the task set. Summing up the fraction of the period spent in each state and multiplying it by the appropriate power consumption measurement produces the overall power consumption for the task set. As a reasonable representation of the load generated by the Linux/idle task, the simulator assumes the Linux/idle task to consume a certain amount of power whose value lies between the power consumptions of a fully loaded and fully idle system running at a given speed. This power value was determined by measuring the power consumption of the laptop with regular Linux running a subset of daemon processes in the background.

We used a single power-state version of LEDF (in effect, EDF) as a comparison point. These tests show the maximum power requirements for the amount of work (computation) to be done. We also used two-speed and three-speed versions of LEDF to observe the effect of adding additional power states. The two-speed version used operating frequencies of 700 and 1100 MHz, and the three-speed version incorporated an intermediate 900 MHz operating frequency. The CPU utilizations ranged from 10 to 80% in increments of 10%. The maximum utilization of 80% was necessary to guarantee that the Linux/idle task had sufficient time available for control operations. Without forcing the scheduler to leave 20% of the period open for the Linux/idle task, the shell became unresponsive, forcing a hard reboot of the machine between each test. We also implemented the cycle-conserving EDF (ccEDF) algorithm of Pillai and Shin [10] and compared our algorithm with it. This implementation of ccEDF uses a set of discrete speeds.

The results are shown in Figure 34.3 for a 15-task task set. Each data point represents the average of three randomly generated task sets for a given utilization value and task set size. LEDF2 (LEDF3) and ccEDF2 (ccEDF3) refer to the use of two (three) processor speeds.

The power savings ranged from 9.4 W in a minimally utilized system to 2.6 W in a fully utilized system. The fully utilized system has lower power consumption under LEDF because LEDF schedules the nonreal-time component at the lowest speed. Note, however, that up to the 50% mark the power savings remain over 9 W and remain in most cases over 7 W for 60% utilization. With a maximum utilization of 80%, the system can still save significant power with a reasonable task load.

A comparison between measured experimental results and simulation results is shown in Figure 34.4. In most cases, the simulated and measured values are the same or within 2% of each other. The simulation results thus provided a very close match to the experimental results, indicating that the simulation engine model accurately models the real hardware. Since the simulation engine does not take into account the scheduler's computation time, the fidelity of the results may degrade for very high task counts due to the extra cost of sorting the deadlines. In order to verify this, we evaluated LEDF with several randomly generated task sets with different utilizations with the number of tasks ranging from 10 to 200 and measured the execution time of the scheduler for each task set. Our results show that the

© 2005 by Chapman & Hall/CRC

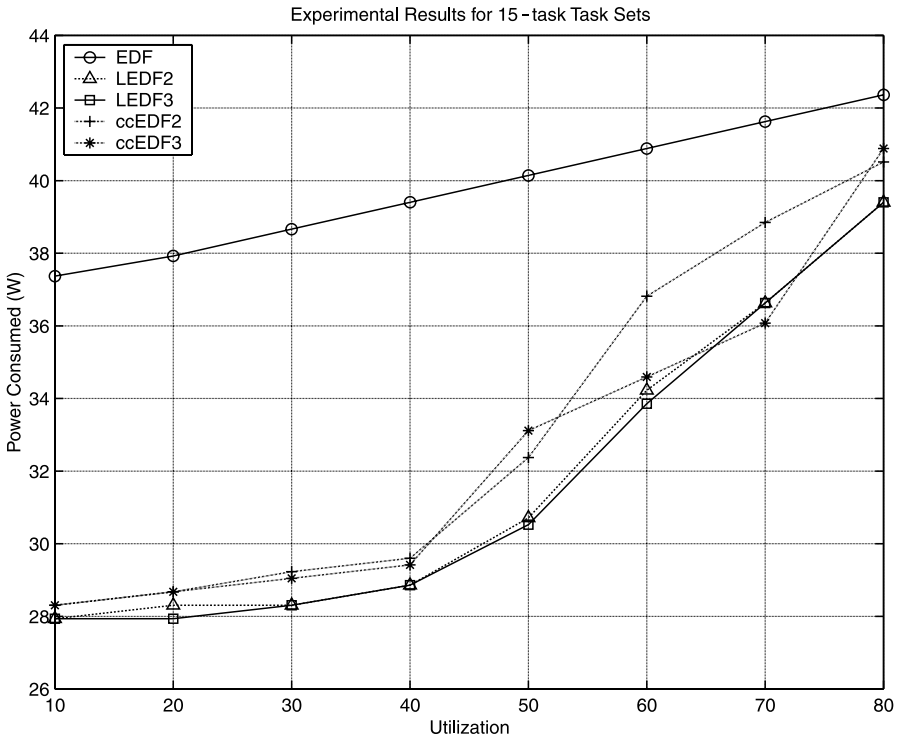


Figure 34.3. Heuristic comparison for 15-task task set.

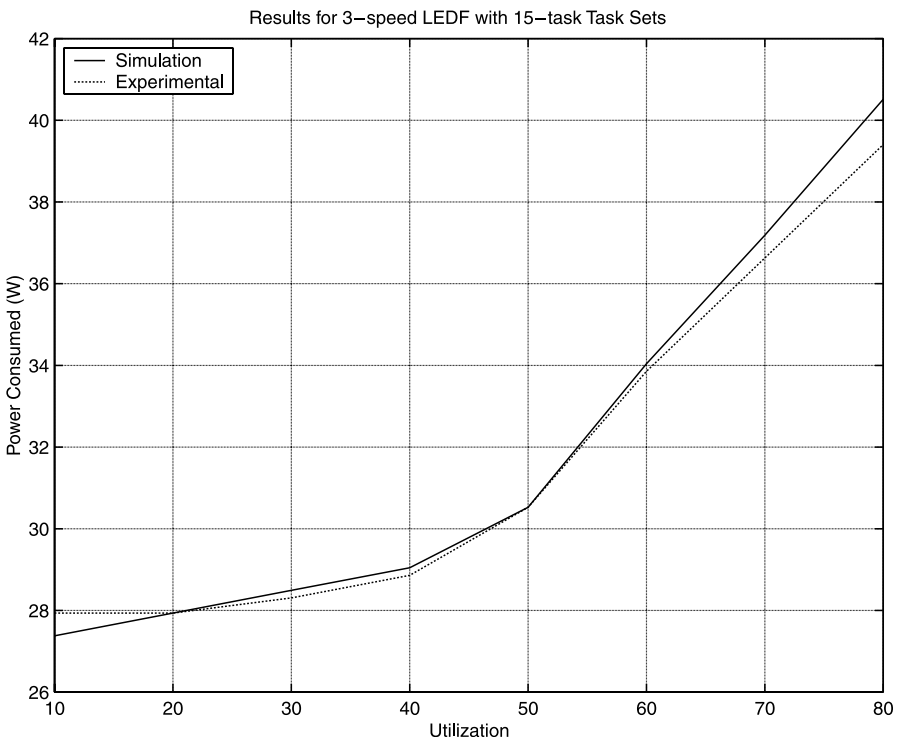


Figure 34.4. Comparison of experimental three-state LEDF with expected results.

© 2005 by Chapman & Hall/CRC

Table 34.3. Measured scheduler overhead for varying task set sizes

Number of tasks	Measured scheduler overhead (ns)
10	1739
20	1824
30	1924
60	3817
120	6621
180	10916
200	12243

execution time of the scheduler was in the order of microseconds, while the task execution times were in the order of milliseconds. For increasing task set size, scheduler runtime increases at a very slow rate. Thus, scheduling overhead does not prove to be too costly for the power-aware version of EDF. For task sets with more than 240 tasks, the RT-Linux platform tended to become unresponsive. These results are shown in Table 34.3. The entries in the table correspond to task sets with 40% utilization, but with varying numbers of tasks. The other task sets we experimented with (task set utilizations of 50 and 80%) also exhibit the same trend in scheduler run time and are not reproduced here. The scheduler overhead in Table 34.3 indicates the time taken by the scheduler to sort the task set and to identify the active task. Even though our implementation of LEDF is currently of $O(n^2)$ complexity and can be replaced by a faster $O(n \log n)$ implementation, it is obvious from Table 34.3 that scheduling overhead is negligible for over a 100 tasks for utilizations ranging from 10 to 80%. For small task sets, where the task set consists of a few hundred tasks, scheduling overhead is negligible compared with task execution times.

34.3 Node-Level I/O-Device-Oriented Energy Management

Prior work on DPM techniques for I/O devices has focused primarily on scheduling devices in non-real-time systems. The focus of these algorithms is minimizing user response times rather than meeting real-time task deadlines; therefore, these methods are not viable candidates for use in real-time systems. Owing to their inherently probabilistic nature, the applicability of the above methods to real-time systems falls short in one important aspect: real-time temporal guarantees cannot be provided. Such methods perform efficiently in interactive systems, where user waiting time is an important design parameter. In real-time systems, minimizing response time of a task does not guarantee that its deadline will be met. It thus becomes apparent that new algorithms that operate in a deterministic manner are needed in order to ensure real-time behavior.

34.3.1 Optimal Device Scheduling for Two-State I/O Devices

In this section, we describe a nonpreemptive optimal offline scheduling algorithm to minimize the energy consumption of the I/O devices in hard real-time systems. In safety-critical applications, offline scheduling is often preferred over priority-based run-time scheduling to achieve high predictability [11]. In such systems, the problem of scheduling tasks for minimum I/O energy can be readily addressed through the technique described here. This algorithm is referred to as the Energy-Optimal Device Scheduler (EDS). For a given job set, EDS determines the start time of each job such that the energy consumption of the I/O devices is minimized, while guaranteeing that no real-time constraint is violated. EDS uses a tree-based branch-and-bound approach to identify these start times. In addition, EDS provides a sequence of states for the I/O devices, referred to as the I/O device schedule, that is provably energy-optimal under hard real-time job deadlines. Temporal and energy-based pruning are used to reduce the search space significantly. Our experimental results show that EDS reduces the energy consumption of I/O devices significantly for hard real-time systems. We next define and describe

© 2005 by Chapman & Hall/CRC

some important terms and assumptions. First, we define the device scheduling problem $\mathcal{P}_{io}$ and describe the task and device models in greater detail.

We are given a task set $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ of n tasks. Each task $\tau_i \in \mathcal{T}$ is defined by (i) an arrival time a_i, (ii) a worst-case execution time c_i, (iii) a period p_i, (iv) a deadline d_i, and (v) a device-usage list L_i. The device-usage list L_i for a task τ_i is defined as the set of I/O devices that are used by τ_i. The hyperperiod H of the task set is defined as the least common multiple of the periods of all tasks. Without loss of generality, we assume that the deadline of a task is equal to its period, i.e. $p_i = d_i$.

A set $\mathcal{K} = \{k_1, k_2, \ldots, k_p\}$ of p I/O devices is used in the system. Each device k_i is characterized by:

- Two power states — a low-power sleep state $ps_{l,i}$ and a high-power working state $ps_{h,i}$
- A wake-up time from $ps_{l,i}$ to $ps_{h,i}$, represented by $t_{wu,i}$
- A shutdown time from $ps_{h,i}$ to $ps_{l,i}$, represented by $t_{sd,i}$
- Power consumed during wake-up $P_{wu,i}$
- Power consumed during shutdown $P_{sd,i}$
- Power consumed in the working state $P_{w,i}$
- Power consumed in the sleep state $P_{s,i}$

Requests can be processed by the devices only in the working state. All I/O devices used by a task must be powered-up before the task starts execution.

In I/O devices, the power consumed by a device in the sleep state is less than the power consumed in the working state, i.e. $P_{s,i} < P_{w,i}$. Without loss of generality, we assume that for a given device k_i, $t_{wu,i} = t_{sd,i} = t_{0,i}$ and $P_{wu,i} = P_{sd,i} = P_{0,i}$. The energy consumed by device k_i is given by

$$E_i = P_{w,i} t_{w,i} + P_{s,i} t_{s,i} + M P_{0,i} t_{0,i}$$

where M is the total number of state transitions for k_i, $t_{w,i}$ is the total time spent by device k_i in the working state, and $t_{s,i}$ is the total time spent in the sleep state.

Incorrectly switching power states can cause increased, rather than decreased, energy consumption for an I/O device. Incorrect switching of I/O devices is eliminated using concept of *breakeven time* [2], which is defined as the time interval for which a device in the powered-up state consumes an energy exactly equal to the energy consumed in shutting a device down, leaving it in the sleep state and then waking it up (Figure 34.5). If any idle time interval for a device is greater than the breakeven time t_{be},

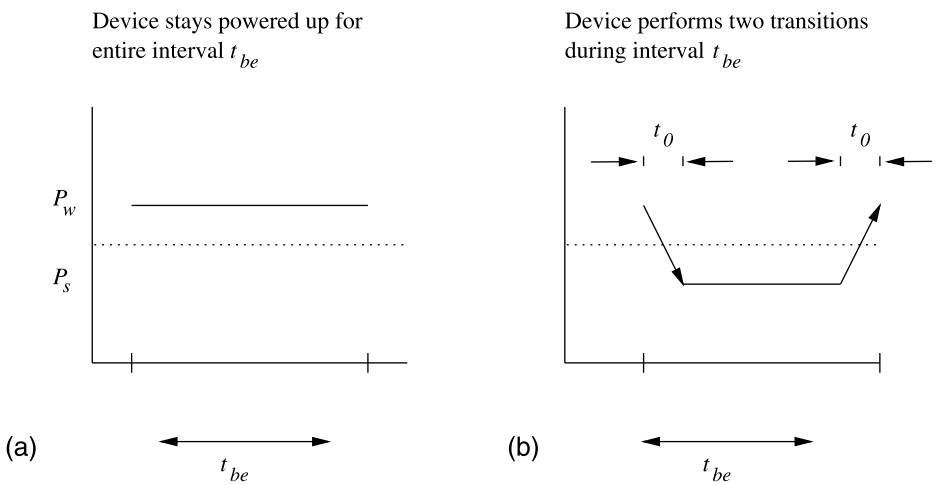


Figure 34.5. Illustration of breakeven time. The time interval for which the energy consumptions are the same in (a) and (b) is called the breakeven time.

© 2005 by Chapman & Hall/CRC

then energy is saved by shutting it down. For idle time periods that are less than the breakeven time, energy is saved by keeping it in the powered-up state.

Associated with each *task set* $\mathcal{T}$ is a *job set* $\mathcal{J} = \{j_1, j_2, \ldots, j_l\}$ consisting of all the instances of each task $\tau_i \in \mathcal{T}$, where $l = \sum_{k=1}^{n} H/p_k$, where H is the hyperperiod and p_k is the period of task τ_k. Except for the period, a job inherits all properties of the task of which it is an instance. This transformation of a pure periodic task set into a job set does not introduce significant overhead because optimal I/O device schedules are generated offline, where scheduler efficiency is not a pressing issue.

For the sake of simplicity, we assume that the devices have only a single sleep state. An extension for devices with multiple low-power states is described in Section 34.3.1.2. In order to ensure that the states of the I/O devices are clearly defined at the completion of the jobs, we assume that the worst-case execution times of the tasks are greater than the transition time of the devices.

The offline device scheduling problem $\mathcal{P}_{\text{io}}$ is formally stated below:

> $\mathcal{P}_{\text{io}}$: Given a job set $\mathcal{J}$ that uses a set $\mathcal{K}$ of I/O devices, identify a set of start times $\mathcal{S} = \{s_1, s_2, \ldots, s_l\}$ for the jobs such that the total energy consumed $\sum_{i=1}^{p} E_i$ by the set $\mathcal{K}$ of I/O devices is minimized and all jobs meet their deadlines.

This set of start times, or schedule, provides a minimum-energy device schedule. Once a task schedule has been determined, a corresponding device schedule is generated by determining the state of each device at the start and completion of each job based on the its device-usage list.

There are no restrictions on the time instants at which device states can be switched. The I/O device schedule that is computed offline is loaded into memory and a timer controls the switching of the I/O devices at run time. Such a scheme can be implemented in systems where tick-driven scheduling is used. We assume that all devices are powered up at time $t = 0$. Next, we describe the theory underlying the EDS algorithm.

34.3.1.1 Pruning Technique

We generate a schedule tree and iteratively prune branches when it can be guaranteed that the optimal solution does not lie along those branches. The schedule tree is pruned based on two factors: time and energy. Temporal pruning is performed when a certain partial schedule of jobs causes a missed deadline deeper in the tree. The second type of pruning — which we call *energy pruning* — is the central idea on which EDS is based. The remainder of this section explains the generation of the schedule tree and the pruning techniques that are employed. We illustrate these through the use of an example.

A vertex v of the tree is represented as a 3-tuple (i, t, e) where i is a job j_i, t is a valid start time for j_i, and e represents the energy consumed by the devices until time t. An edge z connects two vertices (i, t, e) and (k, l, m) if job j_k can be successfully scheduled at time l given that job j_i has been scheduled at time t. A path from the root vertex to any intermediate vertex v has an associated order of jobs that is termed a *partial schedule*. A path from the root vertex to a leaf vertex constitutes a *complete schedule*. A *feasible schedule* is a complete schedule in which no job misses its associated deadline. Every complete schedule is a feasible schedule (temporal pruning eliminates all *infeasible* partial schedules).

An example task set $\mathcal{T}1$ consisting of two tasks is shown in Table 34.4. Each task has an arrival time, a worst-case execution time, and a period. We assume that the deadline for each task is equal to its period. Task τ_1 uses device k_1 and task τ_2 uses device k_2. Table 34.5 lists the instances of the tasks, arranged in

Table 34.4. Example task set $\mathcal{T}1$

Task	Arrival time	Completion time	Period (deadline)	Device-usage list
τ_1	0	1	3	k_1
τ_2	0	2	4	k_2

© 2005 by Chapman & Hall/CRC

Table 34.5. List of jobs for task set $\mathcal{T}1$ from Table 34.4

	j_1	j_2	j_3	j_4	j_5	j_6	j_7
a_i	0	0	3	4	6	8	9
c_i	1	2	1	2	1	2	1
d_i	3	4	6	8	9	12	12

increasing order of arrival. In this example, we assume a working power of 6 units, a sleep power of 1 unit, a transition power of 3 units and a transition time of 1 unit.

We now explain the generation of the schedule tree for the job set shown in Table 34.5. The root vertex of the tree is a dummy vertex. It is represented by the 3-tuple $(0, 0, 0)$ that represents dummy job j_0 scheduled at time $t = 0$ with an energy consumption of 0 units. We next identify all jobs that are released at time $t = 0$. The jobs that are released at $t = 0$ for this example are j_1 and j_2. Job j_1 can be scheduled at times $t = 0$, $t = 1$, and $t = 2$ without missing its deadline. We also compute the energy consumed by all the devices up to times $t = 0$, $t = 1$, and $t = 2$. The energy values are 0 units, 8 units and 10 units respectively (Figure 34.6 explains the energy calculation procedure). We therefore draw edges from the dummy root vertex to vertices $(1, 0, 0)$, $(1, 1, 8)$, and $(1, 2, 10)$. Similarly, job j_2 can be scheduled at times $t = 0$, $t = 1$, and $t = 2$ and the energy values are 0 units, 8 units, and 10 units respectively. Thus, we draw three more edges from the dummy vertex to vertices $(2, 0, 0)$, $(2, 1, 8)$ and $(2, 2, 10)$. Note that job j_2 would miss its deadline if it were scheduled at time $t = 3$ (since it has an execution time of 2 units). Therefore, no edge exists from the dummy node to node $(2, 3, e)$, where e is the energy consumption up to time $t = 3$. Figure 34.7 illustrates the tree after one job has been scheduled. Each level of depth in the tree represents one job being successfully scheduled.

We then proceed to the next level. We examine every vertex at the previous level and determine the jobs that can be scheduled next. By examining node $(1, 0, 0)$ at level 1, we see that job j_1 would complete

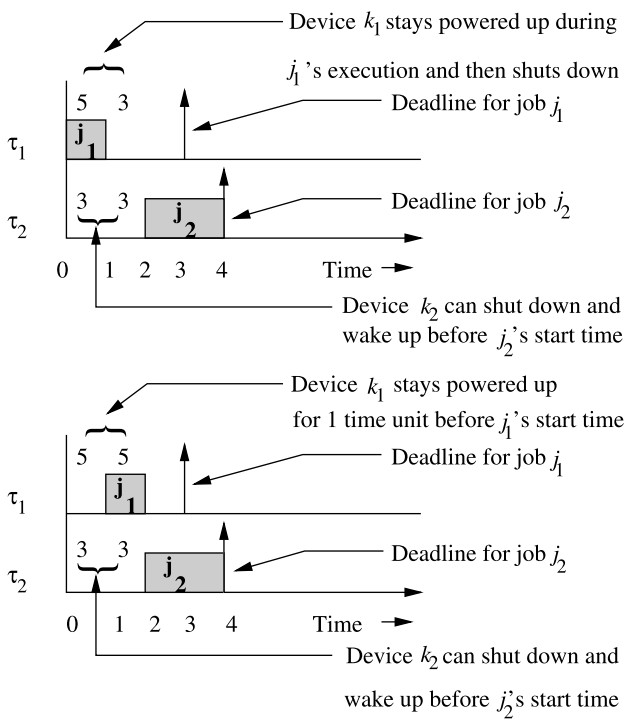


Figure 34.6. Calculation of energy consumption.

© 2005 by Chapman & Hall/CRC

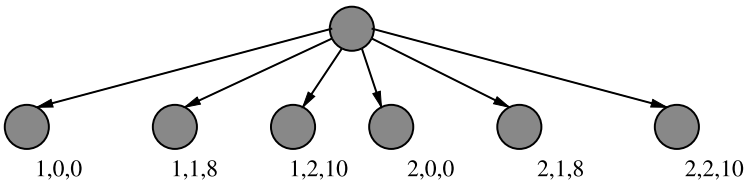


Figure 34.7. Partial schedules after one scheduled job.

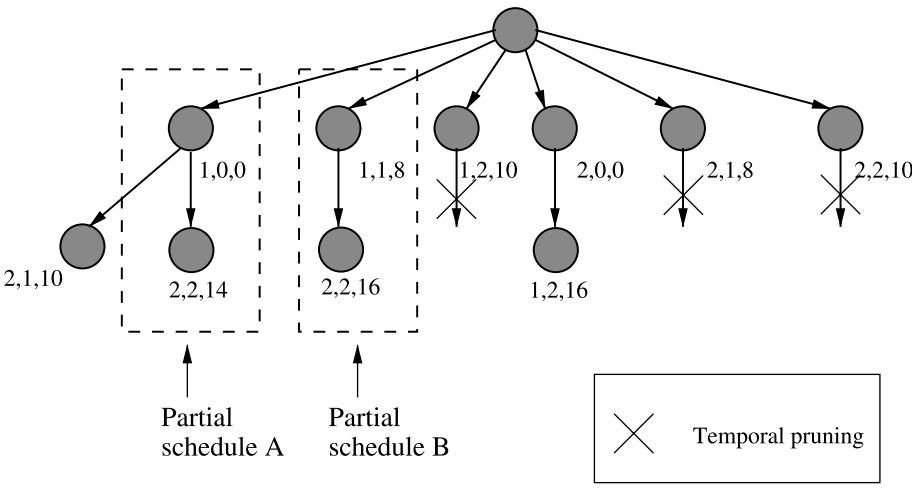


Figure 34.8. Partial schedules after two scheduled jobs.

its execution at time $t = 1$. The only other job that has been released at $t = 1$ is job j_2. Thus, j_2 can be scheduled at times $t = 1$ and $t = 2$ *after* job j_1 has been scheduled at $t = 0$. The energies for these nodes are computed and edges are drawn from $(1, 0, 0)$ to $(2, 1, 10)$ and $(2, 2, 14)$. Similarly, examining vertex $(1, 1, 8)$ results in vertex $(2, 2, 16)$ at level 2. The next vertex at level 1, i.e. vertex $(1, 2, 10)$, results in a missed deadline at level 2. If job j_1 were scheduled at $t = 2$, then it would complete execution at time $t = 3$. The earliest time at which j_2 could be scheduled is $t = 3$; however, even if it were scheduled at $t = 3$, it would miss its deadline. Thus, scheduling j_1 at $t = 2$ does not result in a feasible schedule. This branch can hence be pruned. Similarly, the other nodes at level 1 are examined and the unpruned partial schedules are extended. Figure 34.8 illustrates the schedule tree after two jobs have been scheduled. The edges that have been crossed out represent branches that are not considered due to temporal pruning.

At this point, we note that vertices $(2, 2, 14)$ and $(2, 2, 16)$ represent the same job (j_2) scheduled at the same time $(t = 2)$. However, the energy consumptions for these two vertices are different. This observation leads to the following theorem:

Theorem 34.1. *When two vertices at the same tree depth representing the same job being scheduled at the same time can be reached from the root vertex through two different paths, and the orders of the previously scheduled jobs along the two partial schedules are identical, then the partial schedule with higher energy consumption can be eliminated without losing optimality.*

Proof: Let us call the two partial schedules at a given depth Schedule A and Schedule B, with Schedule A having lower energy consumption than Schedule B. We first note that Schedule B has higher energy consumption than Schedule A because one or more devices have been in the powered-up state for a longer period of time than necessary in Schedule B. Assume that i jobs have been scheduled, with job j_i being the last scheduled job. Since we assume that the execution times of all jobs are greater than the

© 2005 by Chapman & Hall/CRC

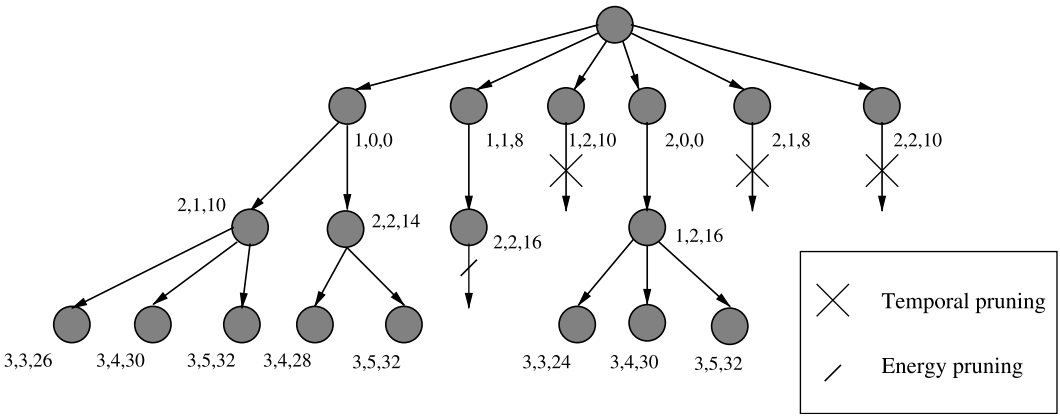


Figure 34.9. Partial schedules after three scheduled jobs.

maximum transition time of the devices, it is easy to see that the state of the devices at the end of job j_i will be identical in both partial schedules. By performing a time translation (mapping the end of job j_i's execution to time $t = 0$), we observe that the resulting schedule trees are identical in both partial schedules. However, all schedules in Schedule B after time translation will have an energy consumption that is greater than their counterparts in Schedule A by an energy value E_δ, where E_δ is the energy difference between Schedules A and B. It is also easy to show that the energy consumed *during* job j_i's execution in Schedule A will always be less than or equal to j_i's execution in Schedule B. This completes the proof of the theorem. □

The application of this theorem to the above example results in partial Schedule B in Figure 34.8 being discarded. As one proceeds deeper down the schedule tree, there are more vertices such that the partial schedules corresponding to the paths to them from the root vertex are identical. It is this "redundancy" that allows for the application of Theorem 34.1, which consequently results in tremendous savings in memory while still ensuring that an energy-optimal schedule is generated. By iteratively performing this sequence of steps (vertex generation, energy calculation, vertex comparison, and pruning), we generate the complete schedule tree for the job set. Figure 34.9 illustrates the partial schedules after three jobs have been scheduled for our example. The complete tree is shown in Figure 34.10. We have not shown paths that have been temporally pruned. The edges that have been crossed out with horizontal slashes represent energy-pruned branches. The energy-optimal device schedule can be identified by tracing the path from the highlighted node to the root vertex in Figure 34.10.

34.3.1.2 The EDS Algorithm

The pseudocode for EDS is shown in Figure 34.11. EDS takes as input a job set $\mathcal{J}$ and generates all possible nonpreemptive minimum energy schedules for the given job set. The algorithm operates as follows. The time counter t is set to zero and openList is initialized to contain only the root vertex $(0, 0, 0)$ (lines 1 and 2). In lines 3 to 10, every vertex in openList is examined and nodes are generated at the succeeding level. Next, the energy consumptions are computed for each of these newly generated vertices (line 11). Lines 15 to 20 correspond to the pruning technique. For every pair of replicated vertices, the partial schedules are checked and the one with the higher energy consumption is discarded. Finally, the remaining vertices in currentList are appended to openList. currentList is then reset. This process is repeated until all the jobs have been scheduled, i.e. the depth of the tree equals the total number of jobs (lines 25 to 28). Note that several schedules can exist with a given energy consumption for a given job set. EDS generates all possible unique schedules with a given energy for a given job set.

© 2005 by Chapman & Hall/CRC

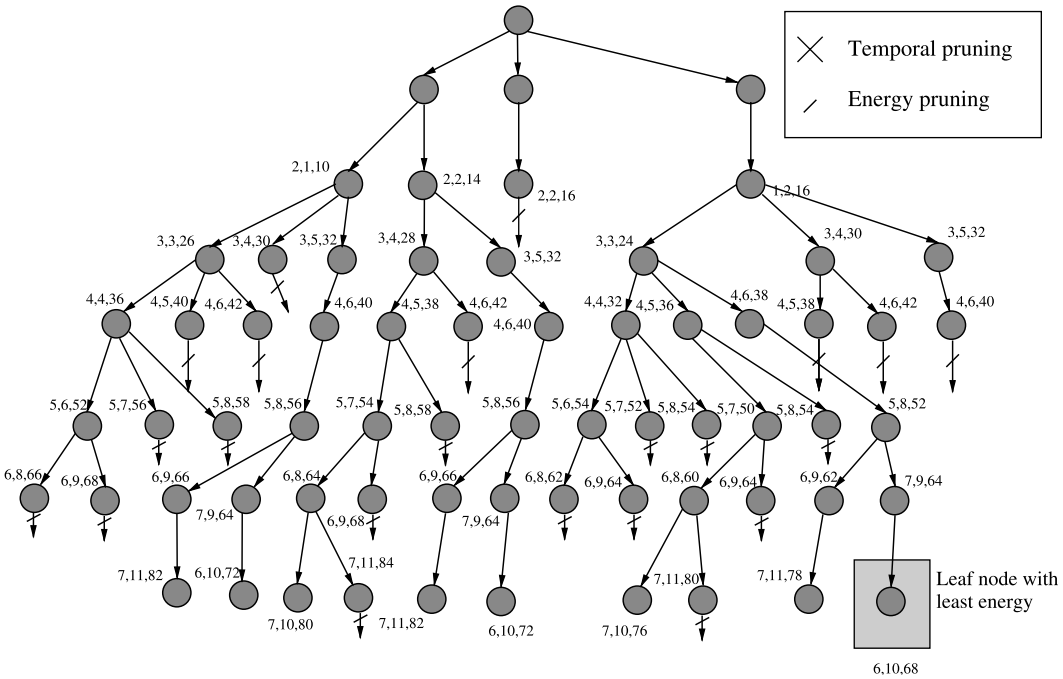


Figure 34.10. Complete schedule tree.

One final comparison of all these unique schedules results in the set of schedules with the absolute minimum energy.

Devices with multiple low-power sleep states can be handled simply by iterating through the list of low-power sleep states and identifying the sleep state that results in the most energy savings for a given idle interval. However, the number of allowed sleep states is limited by our assumption that the transition time from a given low-power sleep state is less than the worst-case completion time of the task.

34.3.1.3 Experimental Results

We evaluated EDS for several periodic task sets with varying hyperperiods and number of jobs. We compare the memory requirement of the tree including the pruning algorithm with the memory requirement of the tree without pruning. Memory requirement is measured in terms of the number of nodes at every level of the schedule tree.

The first experimental task set, shown in Table 34.6, consists of two tasks with a hyperperiod of 20. The device-usage lists for tasks were randomly generated. Expansion of the task set in Table 34.6 results in the job set shown in Table 34.7. Figures 34.12(a) and (b) shows the task and device schedules generated for the task set in Table 34.6 using the fixed-priority rate-monotonic scheduling algorithm [12]. Since device k_3 is used by both tasks, it stays powered up throughout the hyperperiod. The device schedule for k_3 is therefore not shown in Figure 34.12.

If all devices are powered up throughout the hyperperiod, then the energy consumed by the I/O devices for any task schedule is 66 J. Figure 34.13 shows an optimal task schedule generated using EDS. The energy consumption of the optimal task (device) schedule is 44 J, resulting in a 33% reduction in energy consumption.

From Figure 34.12(b), we see that device k_2 stays powered up for almost the entire hyperperiod and device k_1 performs ten transitions over the hyperperiod. Moreover, device k_2 stays powered up even when it is not in use, due to the fact that there is insufficient time for shutting down and powering the device back up. By examining Figure 34.13(a) and (b), we deduce that minimum energy will be

© 2005 by Chapman & Hall/CRC

```
Procedure EDS(𝒥, l)
𝒥: Job set.
l: Number of jobs.
openList: List of unexpanded vertices.
currentList: List of vertices at the current depth.
t: time counter.

1. Set t = 0; Set d = 0;
2. Add vertex (0,0) to openList;
3. for each vertex v = (j_i, time) in openList {
4.     Set t = time + c_i;
5.     Find set of all jobs 𝒥' released up to time t;
6.     for each job j ∈ 𝒥' {
7.         if j has been previously scheduled
8.             continue;
9.         else {
10.            Find all possible scheduling instants for j;
11.            Compute energy for each generated vertex;
12.            Add generated vertices to currentList;
13.        }
14.    }
15.    for each pair of vertices v_1, v_2 in currentList {
16.        if v_1 = v_2 and
               partial schedule(v_1) = partial schedule(v_2) {
17.            if E_v1 > E_v2
18.                Prune v_1;
19.            else
20.                Prune v_2;
21.        }
22.    }
23.    Add unpruned vertices in currentList to openList;
24.    Clear currentList;
25.    Increment d;
26.        If d = l
27.            Terminate.
28. }
```

Figure 34.11. Pseudocode description of EDS.

Table 34.6. Experimental task set $\mathcal{T}1$

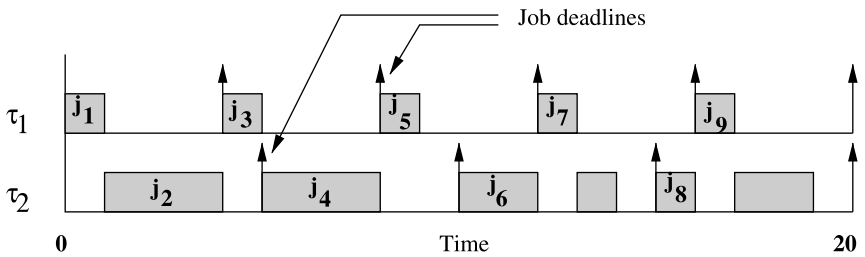
Task	Execution time	Period (deadline)	Device list
τ_1	1	4	k_1, k_3
τ_2	3	5	k_2, k_3

consumed if (i) the time for which the devices are powered up is minimized, (ii) the time for which the devices are shutdown is maximized, and (iii) the number of device transitions is minimized [however, if the transition power of a device k_i is less than its active (operating) power, then energy is minimized by forcing any idle interval for the device to be at least $2t_{0,i}$]. In Figure 34.13(b), no device is powered up when it is not in use. Furthermore, by scheduling jobs of the same task one after the other, the number

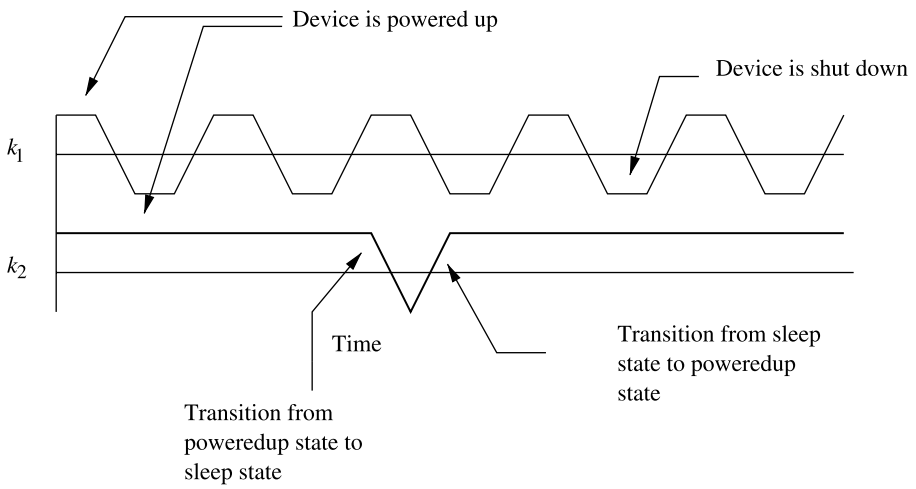
© 2005 by Chapman & Hall/CRC

Table 34.7. Job set corresponding to experimental task set $\mathcal{T}1$

	j_1	j_2	j_3	j_4	j_5	j_6	j_7	j_8	j_9
a_i	0	0	4	5	8	10	12	15	16
c_i	1	3	1	3	1	3	1	3	1
d_i	4	5	8	10	12	15	16	20	20



(a)



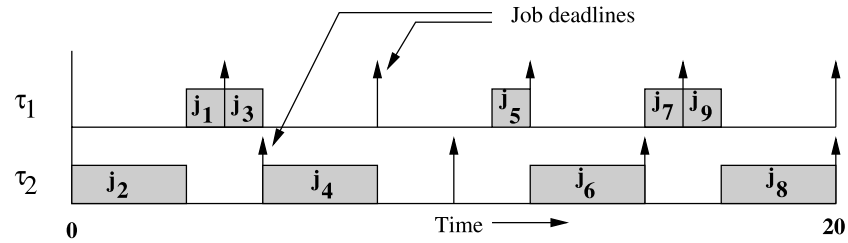
(b)

Figure 34.12. Task schedule for task set in Table 34.6 using rate-monotonic algorithm (RMA).

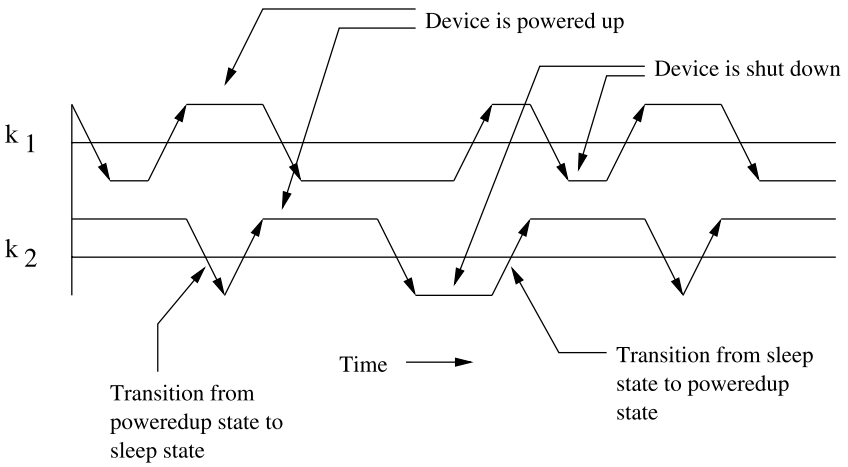
of device transitions is minimized, resulting in the maximization of device sleep time. Our approach to reducing energy consumption is to find jobs with the maximum device-usage overlap and schedule them one after the other. Indeed, two jobs will have maximum overlap with each other if they are instances of the same task. This is the approach that EDS follows.

A side effect of scheduling jobs of the same task one after the other is the maximization of task activation jitter (see Figure 34.13). In some real-time control systems, this is an undesirable feature, which reduces the applicability of EDS in such systems. However, it is clear that jobs of the same task must be scheduled one after the other in order to minimize device energy. Therefore, it appears that scheduling devices for minimum energy and minimizing activation jitter are not always compatible goals.

© 2005 by Chapman & Hall/CRC



(a)



(b)

Figure 34.13. Optimal task schedule for Table 34.6.

In order to illustrate the effectiveness of the pruning technique, we compare EDS with an exhaustive enumeration (EE) method which generates all possible schedules for a given job set. The rapid growth in the state space with EE is evident from Table 34.8. We see that the number of vertices generated by EE is enormous, even for a relatively small task set as in Table 34.6. In contrast, EDS requires far less memory. The total number of vertices for EDS is 87% less than that of EE.

By changing the periods of the tasks in Table 34.6, we generated several job sets whose hyperperiods ranged from $H = 20$ to $H = 40$ with the number of jobs J ranging from 9 to 13. For job sets larger than this, EE failed due to lack of computer memory. EE also took prohibitively large amounts of time to run to completion. These experiments were performed on a 500 MHz Sun workstation with 512 MB of RAM and 2 GB of swap space. The results are shown in Table 34.9.

For job sets with the number of jobs being greater than 17 jobs, the EDS algorithm failed due to insufficient memory. We circumvent this problem by breaking up the vertices generated at level 1 into several separate subproblems. Energy pruning is then performed within and across each subproblem. This is explained in greater detail in the next paragraph.

Let us consider our running example for pruning. Figure 34.7 illustrates the partial schedule tree after one job has been scheduled. The original EDS algorithm expands each of these nodes in a breadth-first fashion and then performs energy-based pruning across all nodes at the second level, as shown in Figure 34.8. At deeper levels, the number of nodes increases tremendously, thereby making excessive demands on memory. An enhancement to EDS that addresses the memory consumption issue is to expand only a single level-1 vertex at a time and perform temporal and energy pruning within this

© 2005 by Chapman & Hall/CRC

Table 34.8. Percentage memory savings

Tree depth i	No. of vertices at depth i		Memory savings (%)
	EE	EDS	
1	7	7	0
2	4	4	0
3	20	14	30
4	18	12	61
5	76	24	68
6	156	26	83
7	270	18	93
8	648	24	96
9	312	8	97
Total	1512	158	90

Table 34.9. Comparison of memory consumption and execution time for EE and EDS[a]

Job set	No. of vertices		Execution time	
	EE	EDS	EE	EDS
$H = 20, J = 9$	1,512	158	<1 s	<1 s
$H = 30, J = 11$	252,931	1,913	2.3 s	<1 s
$H = 35, J = 12$	2,964,093	2,297	28.2 s	4.6 s
$H = 40, J = 13$	23,033,089	4,759	7 m in 15 s	35.2 s
$H = 45, J = 14$	—	7,815	—	2 m in 29.5 s
$H = 55, J = 16$	—	18,945	—	2 h 24 m in 15 s
$H = 60, J = 17$	—	30,191	—	5 h 10 m in 23.2 s

[a]Dashes indicate failed due to insufficient memory.

single subproblem. The memory requirement is therefore reduced significantly. The minimum-energy schedule derived from solving this single subproblem is then recorded. When the next subproblem is solved, energy pruning is performed both within the current subproblem and across all previously solved subproblems. The solution of a single subproblem results in a minimum-energy schedule with a given level-1 job. This energy value is used as an additional bound that is used for further pruning, even at intermediate depths, in succeeding subproblems. With this enhancement, we were able to solve job sets of up to 26 jobs. Even larger problem instances can be solved by breaking the vertices at lower levels into independent subproblems. Here, however, we restrict ourselves only to level-1 subproblems.

The results for the enhanced EDS algorithm are shown in Table 34.10. For this set of experiments, we used a PC running at 1.4 GHz with 512 MB of RAM.

The energy consumptions using EDS are also compared with the case where all devices are powered up. Each job in the job set uses one or more out of three I/O devices whose power values for each are shown in Table 34.11. These values pertain to real devices that are currently deployed in embedded systems. The minimum-energy schedules generated by EDS result in energy savings of up to 45% for the larger job sets listed in the table. The growth of the search space (and corresponding increase in execution time) is also evident from the table. An important point to note here is that the use of the energy value of a complete schedule obtained from solving a single subproblem as a bound results in significant pruning at lower levels in the tree. Therefore, the time taken to search the final set of complete schedules for a minimum-energy schedule is significantly reduced. This results in faster execution times for the enhanced EDS algorithm.

© 2005 by Chapman & Hall/CRC

Table 34.10. Energy consumption using EDS

Job set	Energy consumption (J)		$\Delta E_1(\%) = \frac{E_{eds} - E_{apu}}{E_{apu}} \times 100$	Execution time enhanced EDS
	Enhanced EDS	All powered up		
$H = 20, J = 9$	44.12	66.60	−33.7	<1 s
$H = 30, J = 11$	60.92	96.9	−37.1	<1 s
$H = 35, J = 12$	69.85	113.05	−38.2	<1 s
$H = 40, J = 13$	78.17	129.20	−39.4	<1 s
$H = 45, J = 14$	87.13	145.35	−40.0	<1 s
$H = 55, J = 16$	104.33	177.65	−41.2	<1 s
$H = 60, J = 17$	112.73	193.80	−41.8	3.98 s
$H = 65, J = 18$	121.53	203.95	−40.4	19.15 s
$H = 70, J = 19$	129.93	226.1	−42.5	58.8 s
$H = 80, J = 21$	147.13	258.4	−43.0	7 m in 31 s
$H = 85, J = 22$	156.0	274.0	−43.0	30 m in 45 s
$H = 90, J = 23$	164.33	290.7	−43.4	2 h 39 m in 35 s
$H = 95, J = 24$	170.45	306.85	−44.5	8 h 9 m in 17.3 s
$H = 105, J = 26$	186.23	339.15	−45.0	50 h 0 m in 26.6 s

E_{eds}: energy consumption using EDS; E_{apu}: energy consumption with devices all powered up.

Table 34.11. Device parameters used in the evaluation of LEDES and MUSCLES

Device k_i	Device type	P_w (W)	$P_{sd}^{i,0} = P_{wu}^{i,1}$ (W)	$P_{sd}^{i,1} = P_{wu}^{i,2}$ (W)	$P_{sd}^{i,2} = P_{wu}^{i,3}$ (W)	t_0 (s)	$P_s^{i,1}$ (W)	$P_s^{i,2}$ (W)	$P_s^{i,3}$ (W)
k_1	HDD [13]	2.3	1.5	0.6	0.3	0.6	1.0	0.5	0.2
k_2	NIC [14]	0.3	0.2	0.05	—	0.5	0.1	0.003	—
k_3	DSP [15]	0.63	0.4	0.1	—	0.5	0.25	0.05	—

Finally, we discuss the impact of the assumption that $P_{sd,i} = P_{wu,i}$ and $t_{sd,i} = t_{wu,i}$ on energy consumption. If $P_{sd,i} < P_{wu,i}$ and $t_{sd,i} < t_{wu,i}$, then we can expect to save more energy. If $P_{sd} < P_{wu}$, then devices will not consume as much energy in transitioning between power states. And if $t_{sd} < t_{wu}$, then devices can be powered-down sooner and can stay in the low-power sleep state for longer periods of time. Hence, without the assumption that $P_{sd,i} = P_{wu,i}$ and $t_{sd,i} = t_{wu,i}$, we can obtain greater savings in energy.

As we noted earlier, optimal device scheduling is an $\mathcal{NP}$-complete problem. However, by making a few simplifying assumptions, online polynomial-time algorithms that generate near-optimal solutions can be developed with relative ease. In the next section, we describe two such algorithms. These algorithms schedule the shutdowns and wake-ups of I/O devices such that energy is reduced, and also ensure that no real-time deadlines are missed.

34.3.2 Online Device Scheduling

In this section we describe the Low-Energy Device Schedule (LEDES), a near-optimal, deterministic device-scheduling algorithm for two-state I/O devices. We assume here that the start time for each job is fixed and known *a priori*. Under this assumption, the device scheduling problem $\mathcal{P}_{io}$ is redefined as follows:

- $\mathcal{P}_{io}$: Given the start times $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ of the n tasks in a real-time task set $\mathcal{T}$ that uses a set $\mathcal{K}$ of I/O devices, determine a sequence of sleep/working states for each I/O device $k_i \in \mathcal{K}$ such

© 2005 by Chapman & Hall/CRC

that the total energy consumed $\sum_{i=1}^{p} E_i$ by $\mathcal{K}$ is minimized and all tasks meet their respective deadlines.

In the following sections, we describe the conditions under which device state transitions are allowed to minimize energy and ensure the timely completion of tasks. These conditions are different for different scenarios; the scenarios are dependent on the execution times of the tasks that comprise the task set and the number of sleep states present in a device. We begin by assuming that all task execution times are greater than the maximum transition time among all devices and all devices have only one sleep state. We then show that ensuring timeliness becomes more complex when devices have multiple power states.

One notable advantage of online I/O device scheduling is that online DPM decision-making can exploit underlying hardware features such as buffered reads and writes. A device schedule constructed offline and stored as a table in memory precludes the use of such features due to its inherently deterministic approach. The flexibility of online scheduling enhances the effectiveness of device scheduling.

The need for deterministic I/O device scheduling policies is motivated in detail by Swaminathan and Chakrabarthy [16], who showed that it is not possible to ensure timely completion of tasks without *a priori* knowledge of future device requests. A naive, probabilistic algorithm cannot be used for real-time task sets. We quantify the determinism required to make device scheduling decisions in hard real-time systems through the notion of *look-ahead*, which is a bound on the number of tasks whose device-usage lists must be examined before making a state transition decision, in order to guarantee that no task deadline is missed. Next, we present the LEDES for online scheduling of I/O devices with two power states.

34.3.2.1 Online Scheduling of Two-State Devices: LEDES Algorithm

LEDES assumes that the execution times of all tasks are greater than the transition times of the devices they use. Under this assumption, the amount of look-ahead required before making wake-up decisions to ensure timeliness is easily bounded. We derive this result by presenting the following theorem from [16]:

Theorem 34.2. *Given a task schedule for a set $\mathcal{T}$ of n tasks with completion times $c_1, c_2, \ldots, c_n$, the device utilization for each task, and an I/O device k_l, it is necessary and sufficient to look ahead m tasks to guarantee timeliness, where m is the smallest integer such that $\sum_{i=1}^{m} c_i \geq t_{0,l}$.*

In most practical cases, the completion times of tasks are greater than the transition times $t_{0,i}$ of device k_i. This leads to the following corollary to Theorem 34.2.

Corollary 34.1. *Given a task schedule for a set $\mathcal{T}$ of tasks with completion times $c_1, c_2, \ldots, c_n$, the device utilization for each task, and an I/O device k_j, it is necessary and sufficient to look ahead one task to ensure timeliness if the completion times of all tasks in $\mathcal{T}$ are greater than the transition time $t_{0,j}$ of device k_j.*

The LEDES algorithm operates as follows (also see Figure 34.14). At the start of task τ_i (line 1), devices not used by the next "immediate" tasks τ_i and τ_{i+1} are put in the sleep state (lines 3 and 4). The time difference between the start of τ_{i+1} and the end of τ_i's execution is evaluated and compared with the transition time $t_{0,j}$ to determine whether k_j's wake-up can be guaranteed at τ_i's finish time. If k_j is powered down, then a wake-up decision must be made (line 8). A device must be woken up at s_i if its wake-up cannot be deferred to τ_i's finish time. This is implemented in line 12 and the device is woken up if needed.

If the scheduling instant at which LEDES is invoked is the completion time of τ_i (line 11) and if k_j is powered up (line 12), then it can be shut down only if it can fully enter the powered down state before s_{i+1}, since there may be a need for it to be woken up again. If k_j is in the sleep state (line 15) and is used by τ_{i+1}, then it must be woken up to ensure the timely start of τ_{i+1}. These decisions are made for each device and the entire process repeats at each scheduling instant (although there is no mention of the break-even time in Figure 34.14, an implicit check is made to ensure that the idle period for a given device is always greater than the breakeven time).

© 2005 by Chapman & Hall/CRC

Algorithm LEDES(k_j, τ_i, τ_{i+1})

```
curr: current scheduling instant;
1.    if curr = s_i
2.        if k_j is powered-up
3.            if k_j ∉ L_i ⋃ L_{i+1}
4.                shutdown k_j
5.        if k_j ∈ L_{i+1}
6.            if s_{i+1} − (s_i + c_i) ≥ t_{0,j}
7.                shutdown k_j
8.        else
9.            if k_j ∈ L_{i+1} and s_{i+1} − (s_i + c_i) < t_{0,j}
10.           wakeup k_j
11.   if curr = s_i + c_i
12.       if k_j is powered-up
13.           if k_j ∉ L_{i+1} and s_{i+1} − curr ≥ t_{0,j}
14.               shutdown k_j
15.       else
16.           wakeup k_j
```

Figure 34.14. The LEDES algorithm.

A simple extension to LEDES can efficiently schedule devices that possess multiple sleep states with the ability to switch from any low-power state directly to the working state. Such a device can be viewed as a device with only two power states. Although the transition times from the sleep states to the powered-up state (and *vice versa*) may be different, the correct sleep state to switch a device to is identified simply by performing a series of transition-time checks to verify that there is sufficient time to wake the device up if it is switched to the selected sleep state. However, LEDES cannot make full use of the available sleep states for devices which possess multiple sleep states, but which do *not* possess the ability to jump to any sleep state from the powered-up state.

We next present a more general I/O-centric power management algorithm for hard real-time systems. This algorithm is called the MUlti-State Constrained Low Energy Scheduler (MUSCLES). MUSCLES can also schedule devices which do not have the ability to jump from the powered-up state to any sleep state. Therefore, we assume that, at a device scheduling instant, a device may be switched from one power state to the next higher or lower power state, i.e. only a single transition is possible at any scheduling instant. In the next section, we describe the MUSCLES algorithm in greater detail.

34.3.3 Low-Energy Device Scheduling of Multi-State I/O Devices

In this section we describe the MUSCLES algorithm. The properties of a real-time periodic task remain unchanged from Section 34.3.1. However, I/O device properties now include parameters to describe the different power states. These device properties are restated here for the sake of completeness. Each I/O device $k_i \in \mathcal{K}$ is now characterized by:

- A set $\mathcal{PS}_i = \{\text{ps}_{i,1}, \text{ps}_{i,2}, \ldots, \text{ps}_{i,m}\}$ of m sleep states
- A powered-up state $\text{ps}_{i,\text{u}}$
- Transition time from $\text{ps}_{i,j}$ to $\text{ps}_{i,j-1}$, denoted by $t_{\text{wu}}^{i,j}$
- Transition time from $\text{ps}_{i,j}$ to $\text{ps}_{i,j+1}$, denoted by $t_{\text{sd}}^{i,j}$
- Power consumed during switching up from state $\text{ps}_{i,j}$ to $\text{ps}_{i,j-1}$, denoted by $P_{\text{wu}}^{i,j}$
- Power consumed during switching down from state $\text{ps}_{i,j}$ to $\text{ps}_{i,j+1}$, denoted by $P_{\text{sd}}^{i,j}$
- Power consumed in the working state P_{w}^{i}
- Power consumed in sleep state $\text{ps}_{i,j}$ denoted by $P_{\text{s}}^{i,j}$

© 2005 by Chapman & Hall/CRC

We assume, without loss of generality, that for each device $k_i \in \mathcal{K}$, $t_{\text{wu}}^{i,j+1} = t_{\text{sd}}^{i,j} = t_{0,i}$ and $P_{\text{wu}}^{i,j+1} = P_{\text{sd}}^{i,j} = P_{0,i}$. The total energy E_i consumed by device k_i over the entire hyperperiod is given by

$$E_i = P_{\text{w}}^i t_{\text{w}}^i + \sum_{j=1}^{m} P_{\text{s}}^{i,j} t_{\text{s}}^{i,j} + M P_{0,i} t_{0,i}$$

where M is the number of state transitions, t_{w}^i is the total time spent by the device in the working state, and $t_{\text{s}}^{i,j}$ is the total time spent by the device in sleep state $\text{ps}_{i,j}$.

In order to provide conditions under which devices can be shut down and powered up, we first define a few important terms.

Inter-task time. The inter-task time IT_i for task τ_i is the time interval between the start of task τ_{i+1} and completion of task τ_i. Thus $\text{IT}_i = s_{i+1} - (s_i + c_i)$. There are two scheduling instants associated with a task τ_i, i.e. the start and completion times of τ_i. For minimum-energy device scheduling under real-time constraints, it is not always possible to schedule devices at all scheduling instants. This is formalized using the notion of a valid scheduling instant.

Valid scheduling instant. The completion time of τ_i is defined to be a valid scheduling instant for device k_j if $s_{i+1} - (s_i + c_i) \geq t_{0,j}$. In other words, the completion time of τ_i is a valid scheduling instant if and only if $\text{IT}_i \geq t_{0,j}$. The start time of τ_i is always a valid scheduling instant. Thus, a task τ_i can have either one or two scheduling instants, depending on the magnitude of IT_i relative to the transition time $t_{0,j}$ of a device k_j. Valid scheduling instants are important for energy minimization. Wake-ups can be scheduled at these points to minimize energy and also ensure that real-time requirements are met. Consider the example shown in Figure 34.15. This figure shows two tasks τ_i and τ_{i+1} with the inter-task time $\text{IT}_i < t_{0,j}$. Assume that device k_1 (first used by task τ_{i+2}) is in state $\text{ps}_{1,1}$ at τ_i's completion time $(s_i + c_i)$. If a device were to be woken up at $s_i + c_i$, then it would complete its transition to state $\text{ps}_{1,0}$ only in the middle of τ_{i+1}'s execution and would be in the higher-powered state for the rest of τ_{i+1}'s execution (i.e. until the next scheduling instant). If, on the other hand, the device were to be woken up at s_{i+1}, then we can still ensure that the device is powered up before task τ_{i+2} starts (with the assumption that $c_{i+1} > t_{0,1}$). However, the device stays in the lower powered state until s_i, resulting in greater energy savings. Hence, we see that wake-ups at valid scheduling instants always result in lowered energy consumption. It is always preferable to wake a device up as late as possible in order to utilize the full potential of online device scheduling.

In Section 34.3.1 we showed that a look-ahead of one task is sufficient when devices have only one sleep state. However, a look-ahead of one task is not sufficient when where devices have multiple low-power sleep states. This is clarified through the example shown in Figure 34.16.

Figure 34.16 shows the execution of three tasks τ_1, τ_2, and τ_3. Assume that the start time of τ_1 is the current scheduling instant. Assume that tasks τ_1 and τ_2 do not use device k_2, which is in sleep state $\text{ps}_{2,2}$ at time s_1. An algorithm using a look-ahead of one task, i.e. looking ahead only to task τ_2, would

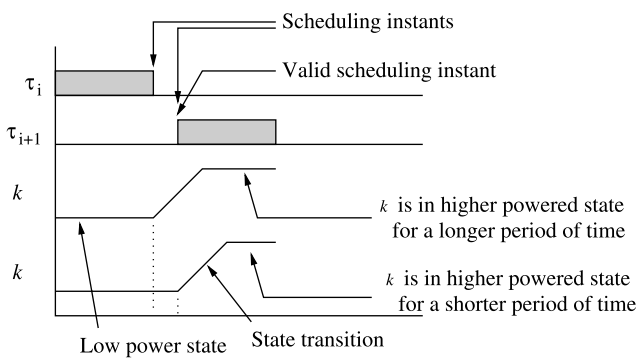


Figure 34.15. Illustration of an invalid scheduling instant.

© 2005 by Chapman & Hall/CRC

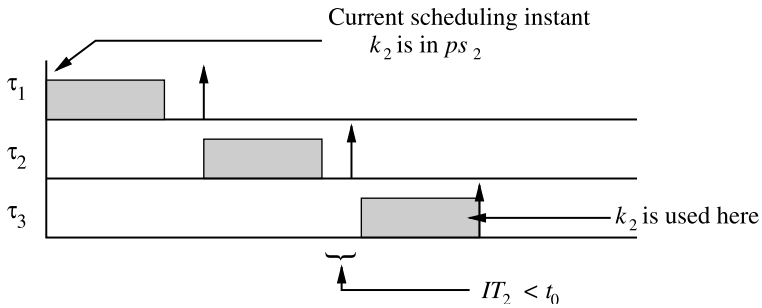


Figure 34.16. To show that look-ahead of one task is insufficient when devices have multiple sleep states.

erroneously decide that there is no need to wake k_2 up at time s_1. The same situation arises at scheduling instant $s_i + c_i$. At τ_2's start time (s_2), looking ahead to task τ_3, k_2 is switched to state $\text{ps}_{2,1}$. At τ_2's completion time, again looking ahead one task to τ_3, k_2 is switched-up to the powered-up state $\text{ps}_{2,\text{u}}$. However, if the inter-task time IT_2 were less than $t_{0,2}$, then k_2 would not have sufficient time to wake up, resulting in τ_3 missing its deadline.

From the above example, it is interesting to note that look-ahead represented as the number of future *tasks* is inadequate for devices with multiple low-power states. When devices have multiple states, look-ahead must be represented as the number of *valid scheduling instants* between tasks. In fact, the notion of look-ahead changes slightly when considering multiple-state I/O devices. Scheduling complexity thus increases with increasing look-ahead due to the additional computational burden of determining look-ahead. Hence, minimizing look-ahead makes the scheduler more efficient.

We now present an upper bound on the look-ahead necessary to ensure timeliness while making shut down decisions for a device [16].

Theorem 34.5. *Consider an ordered set $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ of n tasks that have been scheduled a priori. Let $\mathcal{K} = \{k_1, k_2, \ldots, k_p\}$ be the set of p I/O devices used by the tasks in $\mathcal{T}$. In order to decide whether to switch a device $k_i \in \mathcal{K}$ from state $\text{ps}_{i,j}$ to $\text{ps}_{i,j+1}$ at task τ_c's start or completion time, it is necessary and sufficient to look ahead L tasks, where L is the smallest integer such that the total number of valid scheduling instants associated with the sequence of tasks $\tau_c, \tau_{c+1}, \ldots, \tau_{c+L-1}$ excluding the current scheduling instant is at least equal to $j+1$. The device k_i* **can be** *switched down from $\text{ps}_{i,j}$ to $\text{ps}_{i,j+1}$ if no task τ_t, $c \le t \le c + L - 1$, uses device k_i.*

If the inter-task times of all tasks are less than the transition time $t_{0,j}$ for device k_j, then Theorem 34.3 yields the following corollary.

Corollary 34.2 *Suppose the intertask time IT_i is less than the transition time $t_{0,i}$ for every task $\tau_c \in \mathcal{T}$. In order for a device $k_i \in \mathcal{K}$ to be switched down from state $\text{ps}_{i,j}$ to $\text{ps}_{i,j+1}$ at the start or completion time of task τ_c, it is necessary and sufficient to look ahead $j+1$ tasks to ensure timeliness. Moreover, no task $\tau_t, i \le t \le j$, must use device k_i.*

On the other hand, if the inter-task times for all tasks are greater than or equal to the transition time $t_{0,j}$, Theorem 34.3 leads to the following corollary.

Corollary 34.3. *Suppose the inter-task time IT_i are greater than or equal to the transition time $t_{0,j}$ for every task $\tau_c \in \mathcal{T}$. In order for a device $k_j \in \mathcal{K}$ to be switched down from state $\text{ps}_{i,j}$ to $\text{ps}_{i,j+1}$ at the start or completion time of task τ_c, it is necessary and sufficient to look ahead $\lceil (j+1)/2 \rceil$ tasks to ensure timeliness. Moreover, device k_j must not be used by any task $\tau_t, i \le t \le j$.*

© 2005 by Chapman & Hall/CRC

Look-ahead increases as the depth of the sleep-state increases. We next present an upper bound on look-ahead for making wake up decisions.

Theorem 34.4. *Consider an ordered set $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ of n tasks and a set $\mathcal{K} = \{k_1, k_2, \ldots, k_p\}$ of p devices used by the tasks in $\mathcal{T}$. Suppose the first task after τ_c that uses device k_i is τ_{c+L}. The device $k_i \in \mathcal{K}$* **must be** *switched up from state $\text{ps}_{i,j+1}$ to $\text{ps}_{i,j}$ at the start or completion time of task τ_c if and only if the total number of valid scheduling instants including the current scheduling instant associated with the tasks $\tau_c, \tau_{c+1}, \ldots, \tau_{c+L-1}$ is exactly equal to $j+1$, where L is the look-ahead from the current scheduling instant.*

Theorems 34.3 and 34.4 form the basis of the MUSCLES algorithm, which is described next.

34.3.3.1 Online Scheduling for Multi-State Devices: MUSCLES Algorithm

For a precomputed task schedule, MUSCLES generates a sequence of power states for every device such that energy is minimized. It operates as follows (also see Figure 34.17): let device k_i be in state $\text{ps}_{i,j}$ at scheduling instant s_m. MUSCLES finds the next task τ_L that uses k_i (line 1). A check is then performed to test whether k_i can be switched down to a lower powered state. This is done by ensuring that there are at least $j+1$ valid scheduling instants between the current scheduling instant and τ_L's start time. The presence of $j+1$ valid scheduling instants implies that device k_i can be switched down from state $\text{ps}_{i,j}$ to $\text{ps}_{i,j+1}$ (line 3). The absence of $j+1$ valid scheduling instants precludes the shutting down of k_i to a lower powered state; a check is then performed to test whether the device must be switched up. If exactly j instants are present, then the device must be switched up in order to ensure timeliness (line 4). At the completion of a task τ_m, the same process is repeated. However, an additional check is performed to test if the current scheduling instant is a valid scheduling instant. This is done to minimize energy consumption. If the current scheduling instant is not a valid scheduling instant, then the device is left in the same state until a valid scheduling instant (line 10). MUSCLES guarantees that no task ever misses its deadline.

Algorithm MUSCLES($\mathcal{S}, \mathcal{PS}, k_i$)

`curr`: current scheduling instant;
At s_m:
1. Find first task τ_L that uses device k_i;
2. Compute number of valid scheduling instants X between s_m and τ_L;
3. **if** $X \geq j+1$
4. **switch down** k_i from $ps_{i,j}$ to $ps_{i,j+1}$;
5. **else if** $X = j$
6. **wake up** k_i from $ps_{i,j}$ to $ps_{i,j-1}$;

At $s_m + c_m$:
7. Find first task τ_L that uses device k_i;
8. Compute number of valid scheduling instants X between s_m and τ_L;
9. **if** $X \geq j+1$
10. **switch down** k_i from $ps_{i,j}$ to $ps_{i,j+1}$;
11. **else if** $X = j$ and `curr` is a valid scheduling instant
12. **wake up** k_i from $ps_{i,j}$ to $ps_{i,j-1}$;
13. **else** leave k_i in $ps_{i,j}$.

Figure 34.17. The MUSCLES algorithm.

© 2005 by Chapman & Hall/CRC

LEDES and MUSCLES are both polynomial-time algorithms. MUSCLES has a worst-case complexity of $O(pn^2)$, where p is the number of I/O devices used in the system and n is the number of tasks in the task set, and LEDES is $O(p)$. The complexity increases in MUSCLES because the amount of look-ahead, in terms of valid scheduling instants, for each device must be computed before any state-transition. Nevertheless, the relatively low complexity of MUSCLES makes online device scheduling for low energy and real-time execution feasible.

34.3.4 Experimental Results

We first evaluated LEDES and MUSCLES with several randomly generated task sets with varying utilizations. The task sets consist of six tasks with varying hyperperiods and randomly generated device-usage lists. These task sets are shown in Table 34.12. Since jobs may be preempted, we consider each preempted slice of a job as two jobs with identical device-usage lists. As a result, the number of jobs listed in each task set in Table 34.12 is an approximation. Each task set is scheduled using the rate-monotonic algorithm. The utilization of each task set is varied from 10 to 90% to observe the impact of slack on the energy consumption of the I/O devices.

While evaluating LEDES, we assumed that the single low-power sleep state for the devices corresponded to the highest-powered sleep state of the device. The energy consumptions at different utilizations for task set T_1 is shown in Figure 34.18. Figure 34.19 illustrates the percentage energy savings for each of the task sets obtained from the LEDES algorithm.

A study of Figure 34.18 reveals that the energy consumption using LEDES and MUSCLES increases with increasing utilization. This is because devices are kept powered up for longer periods of time within the hyperperiod. The resulting decrease in sleep time causes this increased energy

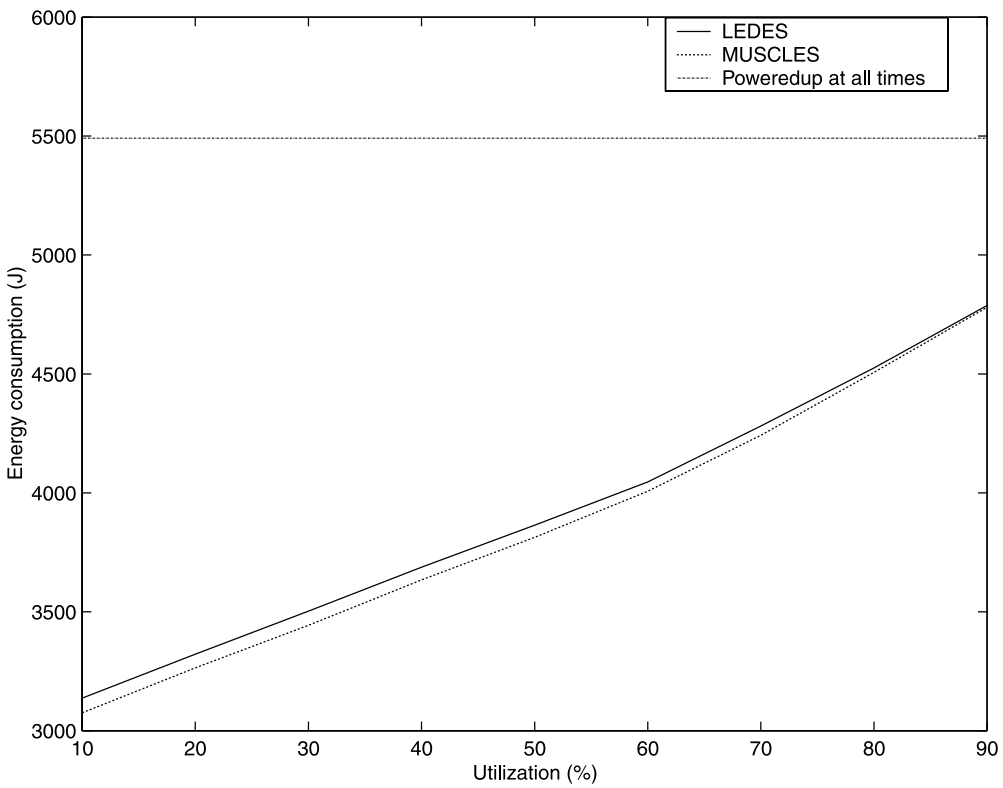


Figure 34.18. Comparison of LEDES and MUSCLES for task set T_1.

© 2005 by Chapman & Hall/CRC

Table 34.12. Evaluation task sets for LEDES and MUSCLES

Task set	Approximate number of jobs	Hyperperiod
T_1	303	1700
T_2	68,951	567,800
T_3	36,591	341,700

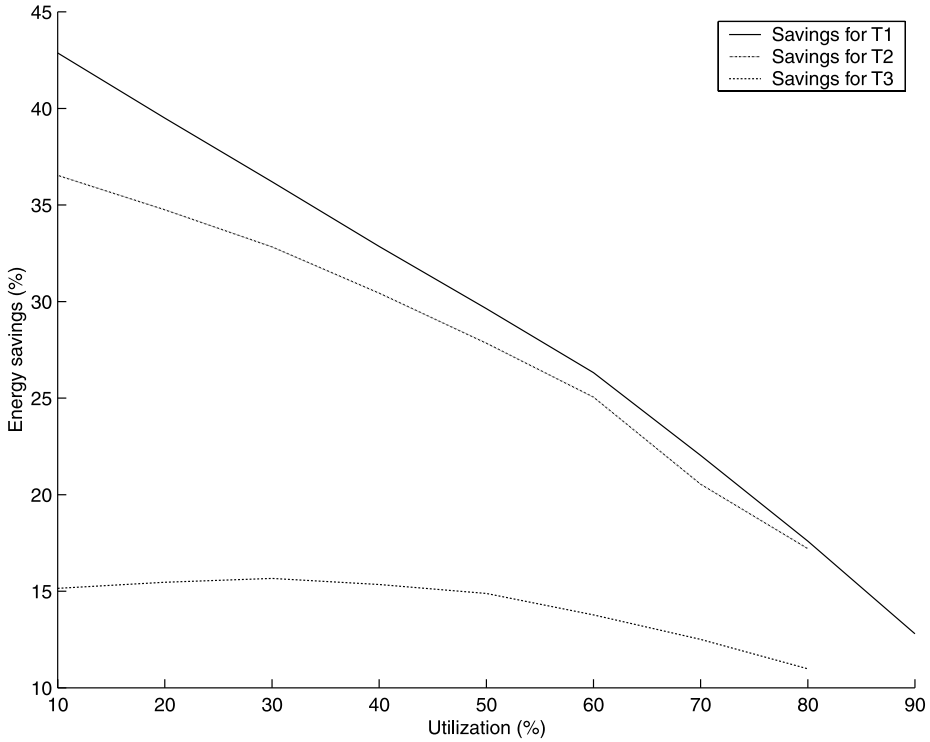


Figure 34.19. Energy savings using LEDES.

consumption. However, we see that energy savings of over 40% can be obtained for task sets with low utilization and over 35% for task sets with high utilization. No task deadlines are missed at any utilization value.

One other important observation that can be made from the graphs is that the savings in energy obtained from MUSCLES *over* LEDES decrease with increasing utilization. This is because the number of valid scheduling instants decreases with increasing utilization. Thus, MUSCLES cannot place devices in deep sleep states as often in high-utilization task sets as it can in low-utilization task sets.

We also evaluated LEDES and MUSCLES with three real-life task sets. These task sets are used in an instrument navigation system (INS) [17], a computer numerical control (CNC) system [18], and an aviation platform (GAP) [19]. The assignment of devices to tasks in the task sets has been inferred from the functionality of the tasks. For example, task 2 in the GAP task set is a communication task that uses the NIC and Task 7 is a status update task that performs occasional reads and writes, and therefore uses a hard disk.

© 2005 by Chapman & Hall/CRC

Table 34.13. Comparison of LEDES and MUSCLES using real-life task sets

Task set	Energy (J)			Savings (%)	
	All powered up	LEDES	MUSCLES	LEDES	MUSCLES
CNC	403,104	197,140	117,604	51	70
INS	16.5×10^6	7.7×10^6	3×10^6	51	81
GAP	381×10^6	210×10^6	153×10^6	45	60

Table 34.14. Comparison of LEDES and EDS

Job set	Energy consumption (J)			$\Delta E_3(\%) = \frac{E_{eds} - E_{ledes}}{E_{ledes}} \times 100$
	EDS	LEDES	Timeout	
$H = 20,\ J = 9$	44.12	59.69	60.21	−26.0
$H = 30,\ J = 11$	60.92	75.29	85.23	−19.0
$H = 35,\ J = 12$	69.85	88.4	100.87	−20.0
$H = 40,\ J = 13$	78.17	102.65	108.76	−23.8
$H = 45,\ J = 14$	87.13	116.9	130.43	−25.4
$H = 55,\ J = 16$	104.33	145.4	155.5	−28.2
$H = 60,\ J = 17$	112.73	159.65	170.43	−29.3
$H = 65,\ J = 18$	121.53	173.9	192.76	−30.1
$H = 70,\ J = 19$	129.93	188.15	216.8	−30.9
$H = 80,\ J = 21$	147.13	216.65	240.98	−31.9
$H = 85,\ J = 22$	156.0	230.9	252.43	−32.4
$H = 90,\ J = 23$	164.33	245.15	270.32	−33.0
$H = 95,\ J = 24$	170.45	259.4	282.53	−34.3
$H = 105,\ J = 26$	186.23	287.9	315.76	−35.4

E_{eds}: energy consumption using EDS; E_{ledes}: energy consumption using LEDES.

Table 34.13 presents the energy consumptions for these task sets using LEDES and MUSCLES. The energy values here are expressed in units of joules, and they correspond to the energy consumption of the I/O devices over the duration of a single hyperperiod. Using LEDES, we obtain an energy saving of 45% for the GAP task set. With MUSCLES, an energy saving of 80% is obtained for the INS task set. Owing to the low utilizations of real-life task sets, significant energy savings can be obtained by intelligently performing state transitions for I/O devices.

Finally, we compare LEDES with EDS and a simple timeout-based scheme. In the timeout-based scheme, a device is powered-down if it has not been used for a prespecified interval of time (here, we assume that the timeout interval is 1 unit). However, a timeout-based scheme cannot be used in hard real-time systems since it cannot guarantee that jobs complete execution before their deadlines. Nevertheless, we compare our algorithms with the timeout method to highlight the effectiveness of our algorithms. These results are presented in Table 34.14. EDS performs better than LEDES and the timeout method for all experimental task sets. Moreover, the timeout method resulted in an average of 6.8 missed job deadlines over all our job sets.

34.4 Conclusions

Energy is an important resource in battery-operated sensor systems. For such systems that operate under real-time constraints, energy consumption must be carefully balanced with real-time responsiveness. In this chapter we have described two approaches to energy minimization in sensor networks: node-level energy minimization and network-level energy minimization.

© 2005 by Chapman & Hall/CRC

The node-level energy minimization techniques described here focus on minimizing the energy consumption of the processor and I/O devices in a sensor node. We described the implementation of a DPM scheme that uses an EDF-based scheduler to support real-time execution. The scheduler is efficient and can be easily integrated into the kernels of real-time OS on sensor nodes. The LEDF algorithm provides significant energy savings in real-time systems.

In many embedded systems, the I/O subsystem is a viable candidate to target for energy reduction. Optimal device schedules for minimum energy consumption can be generated using the offline scheduling algorithm described here. However, the device scheduling problem is known to be $\mathcal{NP}$-complete. With the assumption that device scheduling decisions are made only at task starts and completions, online polynomial-time low-energy I/O device scheduling algorithms to generate near-optimal device schedules can be developed. The first online algorithm described here, called LEDES, efficiently schedules I/O devices that possess two power states: a high-powered working state and a low-powered sleep state. Even under this somewhat restrictive assumption, experimental results show that energy savings of over 40% can be obtained. A generalized version of LEDES, called MUSCLES, that schedules devices with more than two low-power sleep states has also been described here. Experimental case studies for real-life task sets show that energy savings of over 50% can be obtained by targeting the I/O subsystem for power reduction. The amount of energy that can be saved decreases with increasing task set utilization; nevertheless, energy savings of over 40% with these device-scheduling algorithms in high-utilization task sets.

Acknowledgments

The following are reprinted with permission of IEEE: Figures 34.5, 34.15, 34.16, 34.18, 34.19, and Tables 34.12 and 34.13 are taken from [16]. (c) 2003 IEEE; Figures 34.2–34.4 and Tables 34.1 and 34.2 are taken from [20]. (c) 2002 IEEE; Figures 34.6–34.8, and Tables 34.4–34.7 are taken from [21]. (c) 2002 IEEE.

References

[1] Chandrakasan, A.P. and Broderson, R., *Low Power Digital CMOS Design*, Kluwer Academic Publishers, Norwell, MA, 1995.

[2] Hwang, C. and Wu, A.C.-H., A predictive system shutdown method for energy saving of event-driven computation, in *Proceedings of the International Conference on Computer-Aided Design*, 1997, 28.

[3] Srivastava, M.B. *et al.*, Predictive system shutdown and other architectural techniques for energy efficient programmable computation, *IEEE Transactions on VLSI Systems*, 4, 42, 1996.

[4] Buttazzo, G.C., *Hard Real-time Computing Systems: Predictable Scheduling Algorithms and Applications*, Kluwer Academic Publishers, Norwell, MA, 1997.

[5] Li, K. *et al.*, A quantitative analysis of disk drive power management in portable computers, in *Proceedings of the Usenix Winter Conference*, 1994, 279.

[6] Newman, M. and Hong, J., A look at power consumption and performance of the 3Com Palm Pilot., http://guir.cs.berkeley.edu/projects/p6/finalpaper.html.

[7] Jeffay, K. *et al.*, On non-preemptive scheduling of periodic and sporadic tasks with varying execution priority, in *Proceedings of the Real-Time Systems Symposium*, December 1991, 129.

[8] AMD PowerNow! Technology, http://www.amd.com/epa/processors/6.32bitproc/8.amdk6pami/x24267/24267a.pdf

[9] The RT-Linux Operating System, http://www.fsmlabs.com/community/.

[10] Pillai, P. and Shin, K.G., Real-time dynamic voltage scaling for low-power embedded operating systems, in *Proceedings of the Symposium on Operating Systems Principles*, 2001, 89.

[11] Xu, J. and Parnas, D.L., Priority scheduling vs. pre-run-time scheduling, *International Journal of Time-Critical Computing Systems*, 18, 2000.

© 2005 by Chapman & Hall/CRC

[12] Liu, C.L. and Layland, J., Scheduling algorithms for multiprogramming in a hard real-time environment, *Journal of the ACM*, 20, 46, 1973.
[13] Fujitsu MHL2300AT Hard Disk Drive. http://www.fcpa.com/support/hard-drives/technical-data.html.
[14] AMD Am 79C874 NetPHY-1LP Low-Power 10/100 Tx/Rx Ethernet Transceiver Technical Datasheet.
[15] Analog Devices Multiport Internet Gateway Processor, http://www.analog.com/processors/index.html.
[16] Swaminathan, V. and Chakrabarty, K., Energy-conscious, deterministic I/O device scheduling in hard real-time systems, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(7), 847, 2003.
[17] Katcher, D. *et al.*, Engineering and analysis of fixed priority schedulers. *IEEE Transactions on Software Engineering*, 19, 920, 1993.
[18] Kim, N. *et al.*, Visual assessment of a real-time system design: case study on a CNC controller, in *Proceedings of the Real-Time Systems Symposium*, 1996, 300.
[19] Locke, D.C. *et al.*, Building a predicatable avionics platform in Ada: a case study, in *Proceedings of the Real-Time Systems Symposium*, 1991, 181.
[20] Swaminathan, V., Schweizer, C.B., Chakrabarty, K., and Patel, A.A., Experiences in implementing an energy-driven task scheduler in RT-Linus, *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*, 229, 2002.
[21] Swaminathan, V. and Chakrabarty, K., Pruning-based energy-optimal device scheduling in hard real-time systems, *Proceedings of the International Symposium of Hardware/Software Co-Design*, 175, 2002.

© 2005 by Chapman & Hall/CRC

35

An Energy-Aware Approach for Sensor Data Communication

H. Saputra, N. Vijaykrishnan, M. Kandemir, R.R. Brooks, and M.J. Irwin

35.1 Introduction

Distributed sensor networks are envisioned to support various new applications that monitor and interact with the physical world [1–3]. These networks are made of many small interacting nodes that have computing, communication, and sensing abilities. Many sensor network applications in eco-monitoring (forest fires, soil moisture) and disaster tracking (contaminant transport, volcanic plume flow tracking) require the sensors to be deployed in locations that are inaccessible (or expensive to access), requiring the sensor nodes to support wireless communication. Further, the sheer number of sensors deployed in distributed networks (in addition to inaccessibility to some nodes) makes it essential for the individual nodes to operate unattended for long duration. A major limiter to the lifetime of the sensor networks is the limited energy available in these nodes. The limited capacity of the battery pack on these nodes and the inability to replace the batteries in these unattended systems makes it important to conserve the energy consumption in these systems.

Communication is an important factor in determining the energy consumption in distributed sensor applications [4]. Energy consumed by transmitting data wirelessly over 10 to 100 m range has been found to consume as much energy as thousands of operations. Further, the overwhelming volume of sensed information and the need to aggregate data from the different sensed nodes for detecting events of interest leads to a large number of communications across the different nodes. In order to reduce the number of communications, many prior approaches limit the communication to within a set of local nodes called a cluster. These clusters support some local processing of the sensed data and extract some useful information and limit the communication to the nodes outside the cluster to this extracted information. For example in a distributed sensor network that detects a vehicle and its possible route, a node that senses an event communicates with other neighboring nodes in its cluster. The information gathered locally in the cluster is used to calculate the position of an event and its possible route. Then this information is forwarded to the next cluster [5].

The focus of this work is on reducing the energy consumed due to the communication that happens between the nodes in a local cluster. Since local nodes frequently share data to identify events of interest,

© 2005 by Chapman & Hall/CRC

it is important to focus on this task. Owing to their spatial proximity, the nodes in the same cluster often sense the same value because of spatial correlation. For example, the temperature values read by adjacent nodes within the same spatial region tend to be similar. Further, sensed data exhibits temporal locality. For example, the temperature values sensed by sensors deployed in a specific region modulate periodically based on the season and time of day. This work exploits this locality of values transmitted by the nodes within a cluster to reduce the energy consumption. Specifically, instead of transmitting the data values themselves, we maintain a cache of recently transmitted values within the cluster and transmit the index of the cached value whenever the same value needs to be transmitted again. Since transmitting the index of a small cache requires a smaller number of bits to be transmitted than actual values, our technique can provide significant energy savings.

In addition to reducing the energy consumption of communication, the proposed approach also addresses another important issue of providing security for sensed data transmissions. Owing to the use of wireless transmission for communication, sensed data will be vulnerable to eavesdropping and tampering. While encryption of the communication data is one possible option for providing security, encryption/decryption is costly in terms of computational resources in terms of time and energy consumption [6]. In our approach, there is inherent security provided for some of the transmissions limiting the number of encryptions that are required for secure data transmissions. Specifically, a value that has locality is not transmitted as raw data but only as the index of the entry containing that value in the table; this provides security against eavesdropping. Hence, encryption can be limited to only transmitting actual values when the table is established. Note that, in our system, the information of interest that we want to keep secure is the real sensed value. The remaining information, such as the index value, can still be observed. This means someone can still detect whether the node sensed the same data or not. This scenario also happens if we use a traditional cryptography, where the cipher text of the same plain text will be the same when using the same secret key.

We evaluate the proposed approach by modeling the energy consumption of a StrongARM processor-based sensor node. Our experiments utilize synthetic data that exhibit varying degrees of locality in the values transmitted among the sensor nodes in order to model different types of application scenario. Further, we vary the frequency of data transmission and the underlying encryption technique to quantify the energy savings that result from the proposed approach. The results from our evaluation show that the proposed technique reduces both the energy consumed by data transmissions and the energy consumed by encryption required for security.

The rest of this paper is organized as follows. Section 35.2 explains our system and experimental methodology. Section 35.3 describes the proposed communication protocol employed for transmitting the sensed data among the nodes within a single cluster of a distributed sensor system. Experimental results showing the effectiveness of our approach are discussed in Section 35.4. Section 35.5 explains the enhanced communication protocol for spatial locality value. Section 35.6 discusses other related work in optimizing energy consumption and improving security in sensor networks. Finally, we provide conclusions in Section 35.7.

35.2 System Assumptions

Sensor networks consist of hundreds of nodes that are deployed and connected to each other wirelessly. Each node typically consists of sensors, embedded processors, and communication hardware.

Our sensor network consists of 32 clusters and each cluster contains four nodes. Each node in the same cluster is placed within the distance d (500 m). The characteristics of our sensor network are:

- *Sensors.* Each node contains one or more sensors. In this work we assume two kinds of sensor network. The first sensor network is a temperature sensor network which employs temperature sensors. We choose this environment for a network with high value locality in sensed data. The second sensor network is a vehicle tracking sensor network. This sensor network is based on the network built by Brooks [5]. Each node of this network contains of three different sensors: acoustic, PIR, and seismic. This application is representative of low value locality.

© 2005 by Chapman & Hall/CRC

- *Computational resources*. The underlying system is based on a 59 MHz StrongARM processor (SA-1100). Three major consumers of energy due to computations are the processor, memory accesses, and memory leakage. The table used to cache the previous sensed values contains 16 entries, with each entry holding 16 bits of data (i.e. we assume that the size of data generated by a sensor is 16 bits).
- *Communication*. In this work, we use the following energy model adapted from Heinzelman *et al.* [7].

$$E_{\mathrm{Tx}}(k, d) = E_{\mathrm{elec}}k + \varepsilon_{\mathrm{amp}}kd^2$$
$$E_{\mathrm{Rx}} = P_{\mathrm{rec}}T_{\mathrm{on}}$$

where E_{Tx} is the energy consumed when transmitting k bits of data within the distance of d, and E_{Rx} is the energy consumed when receiving/listening for data during T_{on} seconds. E_{elec} is 50 nJ/bit, $\varepsilon_{\mathrm{amp}}$ is 100 pJ/bit/m^2 and P_{rec} is 0.072 mW.

As we can see, the energy consumption due to communication is proportional to the packet size and the square of the distance between two nodes. Our approach tries to reduce the packet size by exploiting the possible value locality that a sensor network might have. We cannot reduce the energy consumption by reducing the distance because we assume that our network is a nonmobile sensor network, meaning that nodes cannot move after they have been deployed. The details of our approach are explained in Section 35.3. We assume that the receiver consumes the same amount of energy when it is either listening (and not receiving data) or receiving data. Further, we shut down the receiver completely during idle periods determined by the message cycle time. Note that our proposed optimization only influences transmission energy and does not change receiver energy.

To protect the confidentiality of the data, the sensor network needs to implement a cryptography algorithm. Different cryptography algorithms consume different amounts of energy due to their different resource and computational requirements. Our approach tries to reduce the number of cryptography processes, both encryption and decryption, by using the index communication (see Section 35.3 for more details).

We use two different cryptography algorithms: one that uses more resources (Rijndael [8–10]) and the other that uses lesser resources (RC5 [11,12]). We simulate those two algorithms on the JouleTrack energy simulator [13] in order to get their energy consumptions when executing on an SA1100 processor. Rijndael encryption consumes on the average 156 μJ and RC5 encryption consumes 29 μJ.

35.3 Caching-Based Communication

The basis of our communication protocol relies on each node in a cluster caching the n previous data values sensed in the cluster. Thus, all the sensor nodes in the cluster must have a coherent cache. If the new sensed value that needs to be transmitted matches with one of the cached values, then only the index number of the matching cache entry is transmitted instead of the actual data value. When there is a cache miss, the node transmits the sensed data along with the index in the cache where this new data should be stored. Since we can have two different kinds of packet based on whether there is a hit or miss in the cache, the transmitted packet contains an additional bit to distinguish the packet type. The packet size when a cache miss occurs is given by

$$\text{Packet Size}_{\mathrm{miss}} = 1 + k + \log_2 n$$

where k is the size of data generated by the sensors and n is the number of entries in the cache. For instance: when we have 16-bit data values and a 16-entry cache, a cache miss results in the transmission of 21 bits data (1 bit to indicate the packet type, 16 bits for the sensed value itself, and 4 bits for the cache index). The packet size transmitted in the case of a cache hit is given by

$$\text{Packet Size}_{\mathrm{hit}} = 1 + \log_2 n$$

© 2005 by Chapman & Hall/CRC

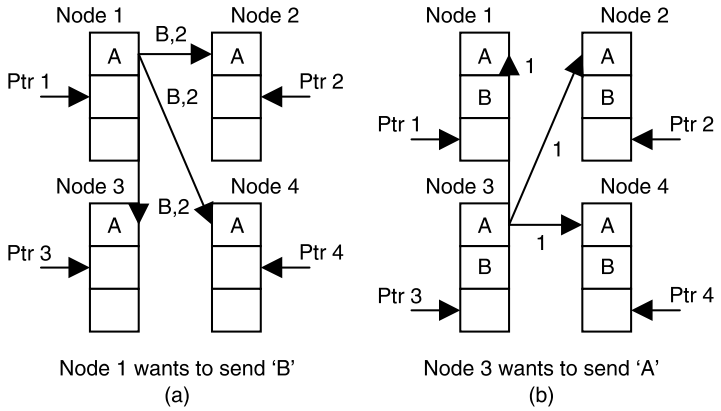


Figure 35.1. Normal communication scenario.

So, when there is a hit, we can reduce the packet size from 16 bits (without using caching) to 5 bits for the example considered above.

Figure 35.1 shows how our caching scheme works in the normal scenario, assuming that there is no communication lost. Nodes 1, 2, 3, and 4 are in the same cluster; so, whenever one of these nodes senses an interesting event it needs to send it to its neighboring nodes. Ptr_1, Ptr_2, Ptr_3, and Ptr_4 are the write pointers for the caches in nodes 1, 2, 3, and 4 respectively. After writing data into a cache, this pointer is changed appropriately to point to the next available free entry. Also note that we reclaim the oldest cache entries to provide the free entries when the cache becomes full. This will be explained further later.

Let us now consider an example. Initially, assume that there are data "A" in each node's cache. Then node 1 is ready to transmit a new sensed data "B." Since node 1 cannot find "B" in its cache (cache miss), it needs to send the value "B" along with the value of Ptr_1 (which has a value of 2) as shown in Figure 35.1(a). Then the other nodes in the cluster receive this data and update their caches. Now all the caches contain both "A" and "B," and the write pointers are changed to point to the third location. Next, node 3 senses a new data value "A." Node 3 will generate a cache hit (because "A" is already in its cache), so it does not need to send "A" again, instead, it will send the pointer that points the location of "A" within its cache (location: 1). So node 3 will send a packet that only contains the index, as shown in Figure 35.1(b).

Since wireless networks can experience packet loss during transmissions, it is important to consider cases where packets are lost. Let us consider the scenario shown in Figure 35.2. Assume that node 1

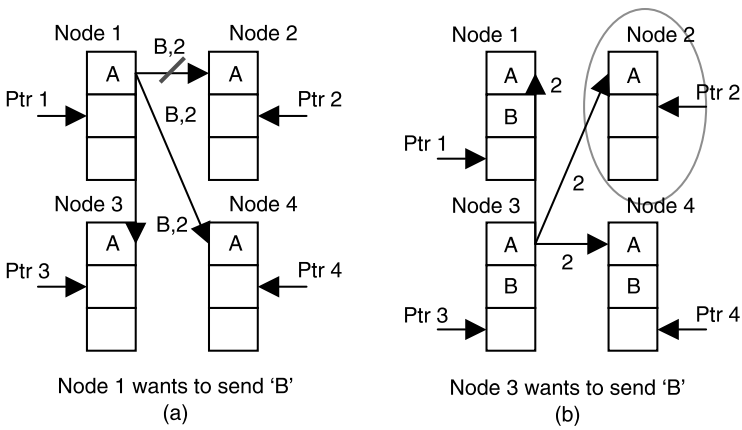


Figure 35.2. Lost communication, Scenario A.

© 2005 by Chapman & Hall/CRC

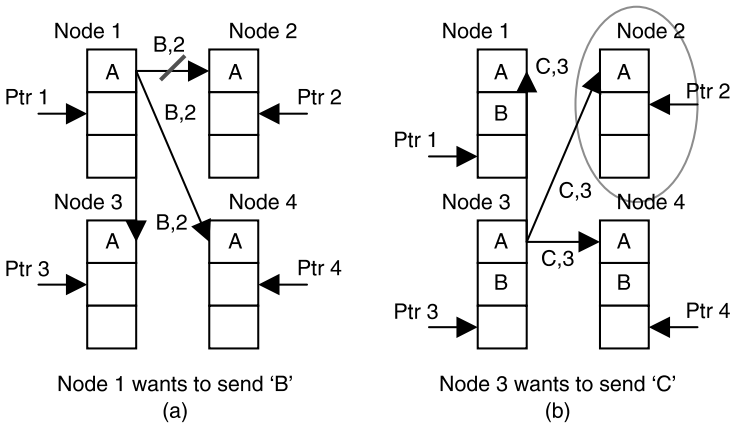


Figure 35.3. Lost communication, Scenario B.

wants to send "B" to nodes 2, 3, and 4 [Figure 35.2(a)]. Nodes 3 and 4 receive this packet, but node 2 does not receive it due to a communication loss. Write pointers of nodes 1, 3, and 4 are changed because they have written new data, but the write pointer of node 2 still points to the second location. Suppose node 3 wants to send "B" to the other nodes, it will send the index 2 (cache hit). Node 2 will detect that its cache is not coherent with the other nodes, caches, because the location 2 of its cache is an invalid entry. Then node 2 updates the missing cache entries by making a request to node 3.

Another lost communication scenario is shown in Figure 35.3. The difference between this scenario and the previous scenario (Figure 35.2) is that node 3 wants to send different data "C" that is not already in its cache. As shown in Figure 35.3(a), node 3 will send a packet that contains both "C" and its Ptr_3 (C, 3). Nodes 1 and 4 will receive this packet normally. On the other hand, node 2 will raise an exception because the index received from the packet does not match with its write pointer (Ptr_2). Then node 2 updates the missing cache entries by making a request to node 3.

Assume now that we have a scenario as represented in Figure 35.3, but now node 2 wants to send data "C" instead of node 3; see Figure 35.4. In this scenario, node 2 will send a packet that contains "C" and its Ptr_2; in this case the value of the packet will be (C, 2). Then nodes 1, 3, and 4 will receive it and raise an error because the index contained in the packet is not the same as the write pointers of these nodes. In this case, all the cache values are flushed.

If we want to cache n previous data values, then we should provide a cache that contains more than n entries, because if we have only n cache lines this can lead to the problem shown in Figure 35.5. In

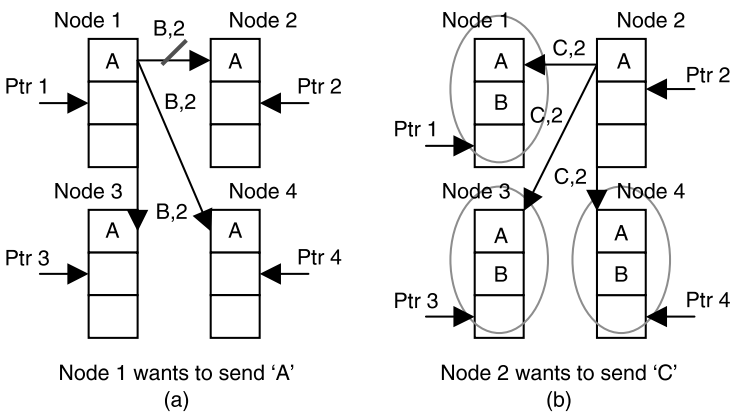


Figure 35.4. Lost communication, Scenario C.

© 2005 by Chapman & Hall/CRC

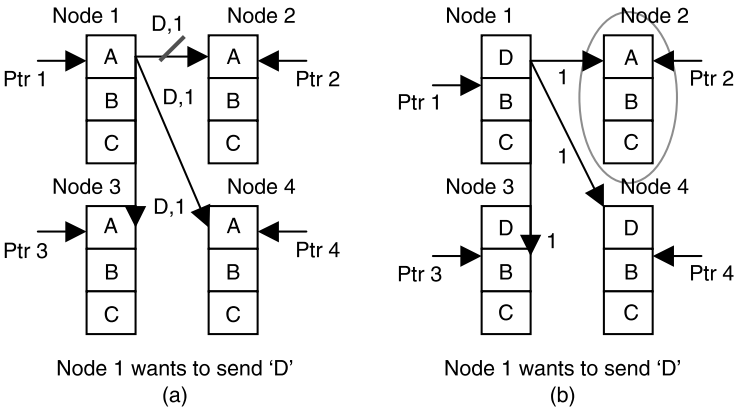


Figure 35.5. Incorrect index interpretation problem.

Figure 35.5, initially all caches are full (so that all write pointers will point to the top locations of caches) and node 1 wants to send data "D." Because that data are not in the cache, node 1 will create a packet that contains "D" and Ptr_1 (D, 1) and send it to the other nodes. Assume that the packet does not reach node 2, then the write pointer of node 2 will not advance to the second location. Nodes 3 and 4 will receive it normally and update their write pointers. Next, if node 1 senses the value "D," it then transmits the index 1, as shown in Figure 35.5(b). Nodes 3 and 4 will look into their cache entry at location 1 and retrieve the correct value "D." However, node 2 will retrieve the wrong value "A." This happens because the value at location 1 of node 2's cache is valid.

To overcome this kind of problem, we use caches that have more cache lines than the actual number of unique values that we want to store ($n+$ caches). Specifically, if n is the number of unique values that we want to store, then we provide m additional cache lines so that the caches will have at least m empty cache lines at any time. Whenever there are n valid entries in the cache, the addition of a new entry results in the oldest entry in the cache being marked invalid. This provides m empty cache lines at all times; thus, unless m communications to a node are lost, the kind of problem illustrated in Figure 35.5 will not occur. Figure 35.6 shows the scenario when we have communication lost on the $n+$ caches. Node 2 in Figure 35.6 knows that its cache is not coherent with the other caches because location 1 in its cache has an invalid entry.

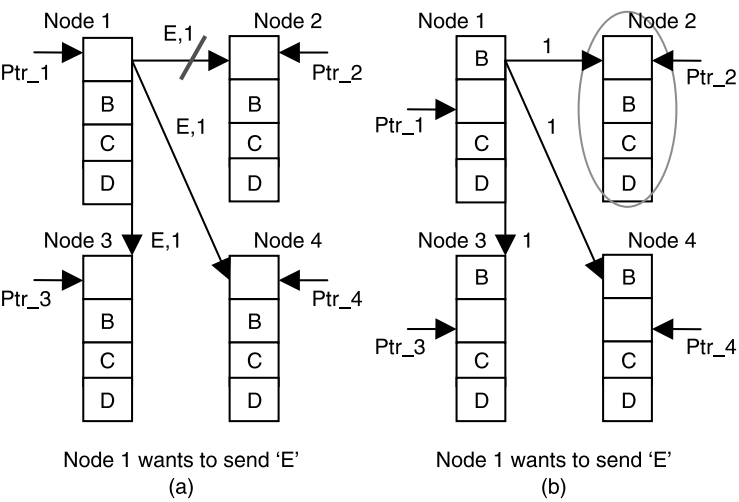


Figure 35.6. Packet lost when using $n+$ cache lines.

© 2005 by Chapman & Hall/CRC

Since we only send the index value whenever we have cache hit, we do not need to apply an encryption process to that packet. We only apply an encryption whenever we have to send the original data value. This implies that the number of cryptography processes needed is less than the original approach.

35.4 Experimental Results

Our experiments utilize the configuration and synthetic data generated shown in Table 35.1. We generated different sets of data to vary the degree of locality in the values transmitted among the sensor nodes in order to model different types of application scenario. While the sensor nodes that measure the temperature in the same cluster exhibit good value locality, other applications, such as vehicle tracking, exhibit poor value locality [5]. Further, we vary the gap between successive data transmissions (called the message cycle) between the different nodes. This, again, is a function of the application and can also vary during the course of an application. For example, the temperature measurements can be more frequent during the summer season, when forest fires are more likely, as opposed to the winter season. We also model different data loss probabilities to see how robust our energy savings are in the presence of varying degrees of data loss. The data loss parameter measures the percentage of transmissions where at least one of the cluster nodes does not receive the data. Finally, we used two different security algorithms, namely RC5 and Rijndael, in our evaluation.

First, we evaluated the energy savings of our approach by varying the value locality assuming a perfect wireless channel as shown in Figure 35.7. Here, a value locality of 10% means that a new sensed value has a 10% chance of hitting in the value cache. We can observe from Figure 35.7 that we can reduce the

Table 35.1. Synthetic data configurations

Processor type	StrongARM SA-1100
Processor speed	59 Mhz
Num of sensors	128 sensors
Num of clusters	32 clusters
Message size	16 bits
Table size	32 bytes
Lost rate	Vary
Value locality	Vary
Message cycle time	Vary
Distance	500 m

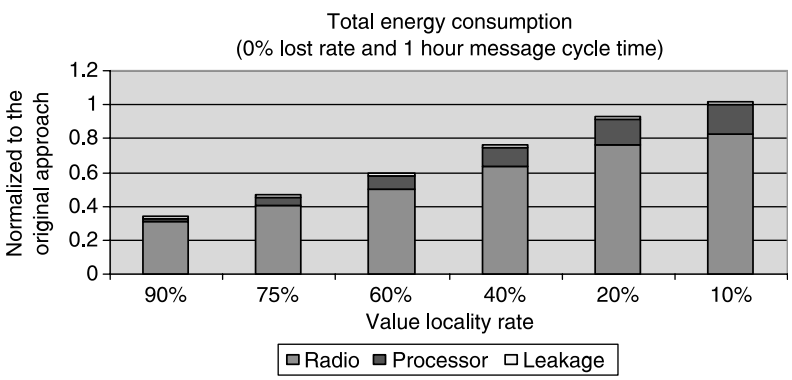


Figure 35.7. Energy consumption based on various value localities using RC5 cryptography. Original approach refers to transmitting data values always.

© 2005 by Chapman & Hall/CRC

energy consumption by more than 60% when we have high locality of 90%. When the locality is only 20%, we get less than 10% reduction in energy consumption. The energy savings accrue from the fewer bits communicated on cache hits and the fewer encryption/decryption employed in the nodes. We also account for the overhead spent in the additional bits for sending the index bits as well as the overhead for accessing the small tables in our measurements. The tables will also consume leakage energy immaterial of whether they are accessed or not, but this leakage is reduced to minimum by applying leakage control techniques during the idle time between messages [14]. Specifically, we observe that the communication energy alone reduces by 60% when the value locality is 90%. In contrast, the communication energy increases by 7% when value locality is only 10%. In order to illustrate the energy savings that accrue from the reduction in the encryption process, Figure 35.8 shows how many encryption processes need to be executed for various value localities and data lost rates.

Figure 35.9 presents how the data lost rate affects the energy consumption. This shows that high data lost rates of more than 20% will make it difficult to maintain coherent value caches and hence may increase the energy consumption when value locality is poor. The results show that we can gain 4% energy reduction even with a 20% data lost rate if we can have 50% value locality. Obviously, our scheme works best when the value locality is high and data lost rate is low. Further, it is still a useful technique for reducing energy even with moderate value locality and less than 20% data lost rates.

Figure 35.10 shows how the message cycle time affects the total energy consumption of the sensor. Since the memory cells that store the cached value consume leakage energy, the energy expended in these cells increases in proportion to the duration of their storage. Hence, the message cycle time is an important factor that determines the relative magnitude of energies consumed by the transmission of

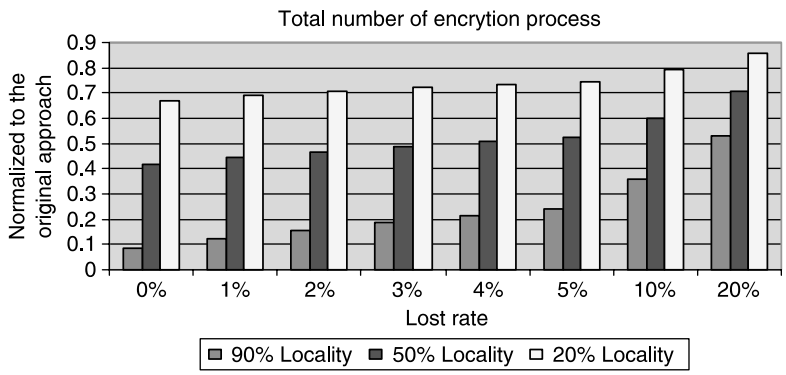


Figure 35.8. Number of encryption processes executed for various value localities and lost rates.

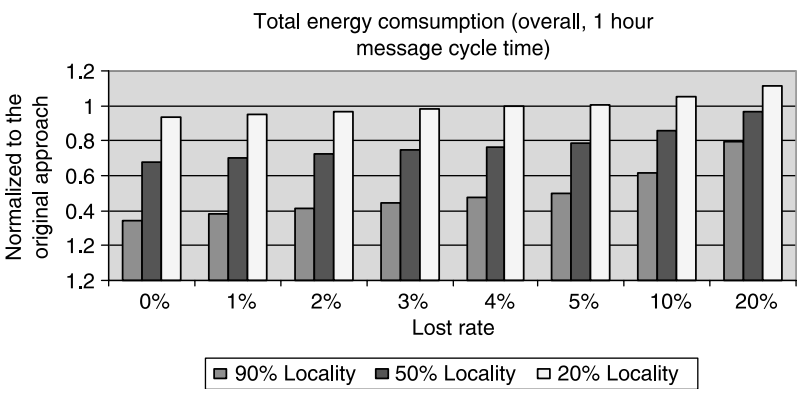


Figure 35.9. Energy consumption based on various lost rates and value locality using RC5 cryptography.

© 2005 by Chapman & Hall/CRC

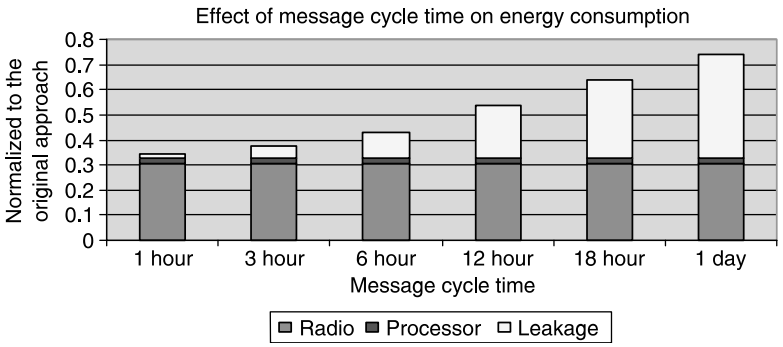


Figure 35.10. Energy consumption based on various message cycle times using 90% value locality and 0% lost rate.

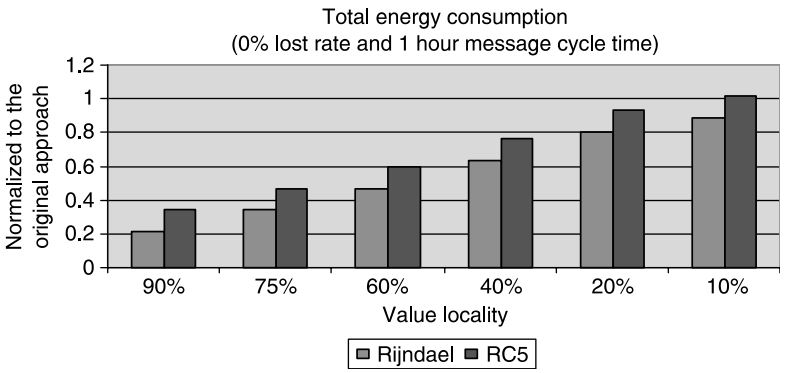


Figure 35.11. Energy consumption based on various value localities using Rijndael and RC5 algorithms.

sensed values and that expended by leakage when caching the values in the sensor nodes. Note that reducing the leakage by employing leakage control techniques [14] does not completely eliminate leakage energy, as data should be retained. From the results in Figure 35.10, we see that the energy reduction obtained from the communication energy reduction is greater than the overhead incurred by caching the data values for the different message cycle times ranging from 1 h to 1 day. We reduce about 25% energy consumption even with a one day message cycle time assuming that we have 90% value locality.

All of the results reported till now are based on the use of the RC5 cryptography algorithm for securing transmissions. The effectiveness of our scheme also depends on the complexity of the security algorithm. When the more complex algorithm Rijndael is employed, we observe more savings, as illustrated in Figure 35.11. Figure 35.12 shows the comparative energy savings for various data lost rates and 50% value locality when using Rijndael and RC5 algorithms. When using the Rijndael algorithm, we obtain around 20% energy savings even with 20% lost rate. On the other hand, the RC5 algorithm only gives us 3% energy savings for the same configuration.

35.5 Spatial Locality

In previous sections we use an index to send the same values that are already in the cache. In this section, we improve our scheme by introducing another packet format that is used to send a data value that is closely related to the latest one (spatial locality). If the absolute difference between the new sensed value and the previous value is less than some threshold, then we send the difference instead of the new value itself. For instance, we use the threshold of 2^8 with the original data size of 16 bits.

© 2005 by Chapman & Hall/CRC

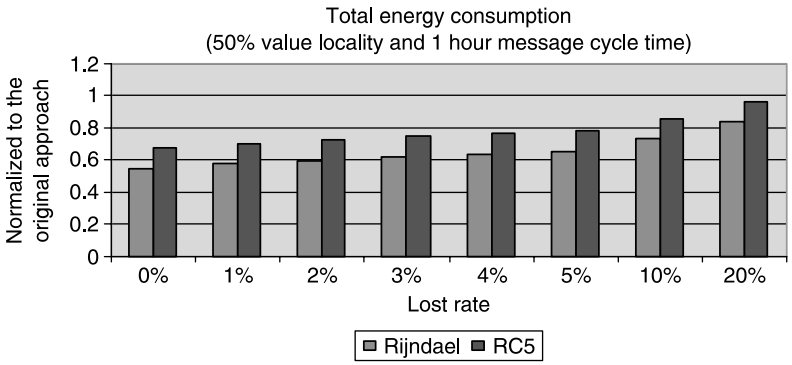


Figure 35.12. Energy consumption based on various lost rates using Rijndael and RC5 algorithms.

This means that if the new sensed value V_{new} is in the range of $(V_{prev} - 2^8 + 1)$ and $(V_{prev} + 2^8 - 1)$, where V_{prev} is the previous sensed value, we send the value difference $(V_{new} - V_{prev})$ instead of V_{new} in a smaller packet. Consequently, the overhead bits needed to identify what type of packet that is being sent requires 2 bits, for instance 00 for the original value packet, 01 for the index packet, and 10/11 for the value difference packet, one for a positive offset and the other for a negative offset. The size of these packets is

$$\text{Packet Size}_{\text{value}} = 2 + k + \log_2 n$$
$$\text{Packet Size}_{\text{index}} = 2 + \log_2 n$$
$$\text{Packet Size}_{\text{diff}} = 2 + \text{Threshold bits} + \log_2 n$$

As we can see from these equations, the additional packet also requires an index value ($\log_2 n$) since it is pointing the location where the new value will be stored. Note that this method is trying to exploit the possible spatial locality. Consequently, if the spatial locality of an application is low, then the energy consumption increases rather than decreases due to the additional bit in the value packet and the index packet, compared with the temporal locality exploitation scheme. Another issue is the size of the threshold bits. This affects the potential for exploiting the spatial locality of an application. When we have a higher threshold, this means that more values will be considered as a value-difference packet since the range is larger. However, a large threshold also requires a larger packet size for the value-difference packet.

The architecture of this approach is similar to the previous one. One cell of the cache is used to store the latest sensed value that is used for the value-difference packet. We can eliminate this additional storage overhead by adding a new pointer in the cache to point to the latest sensed or received value, as shown in Figure 35.13.

The latest sensed value in Figure 35.13 is pointed to by the "latest" pointers. Figure 35.13(a) shows the original cache entries with the latest value of A. Node 1 then sends a new sensed value, B. The latest pointer now points to the value of B, as shown in the Figure 35.13(b). Then node 3 sends an index value 1 (since it senses the already known value, i.e. A). After the other nodes receive this index value, their latest pointers will be updated so that they will point to the value of A, as shown in Figure 35.13(c).

Figure 35.14 shows the situation when node 3 from Figure 35.13 sends a new value using a value-difference packet, i.e. node 3 sends a packet with the offset value of +C. After receiving this packet, the other nodes will store a new value D in the third location, where D is the sum of the value pointed by the latest pointer, B, and the offset (+C).

The situation when there is a packet loss is similar to the method we use in the temporal locality exploitation. The difference is that when the current sensed value is re-sent using the normal packet, the latest pointers are updated to point to this index.

© 2005 by Chapman & Hall/CRC

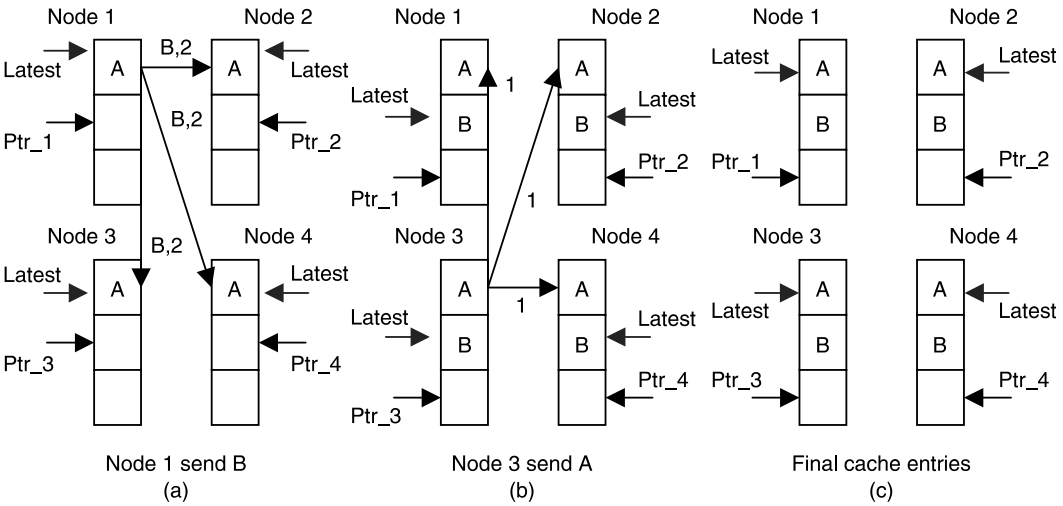


Figure 35.13. Spatial value locality using additional pointer.

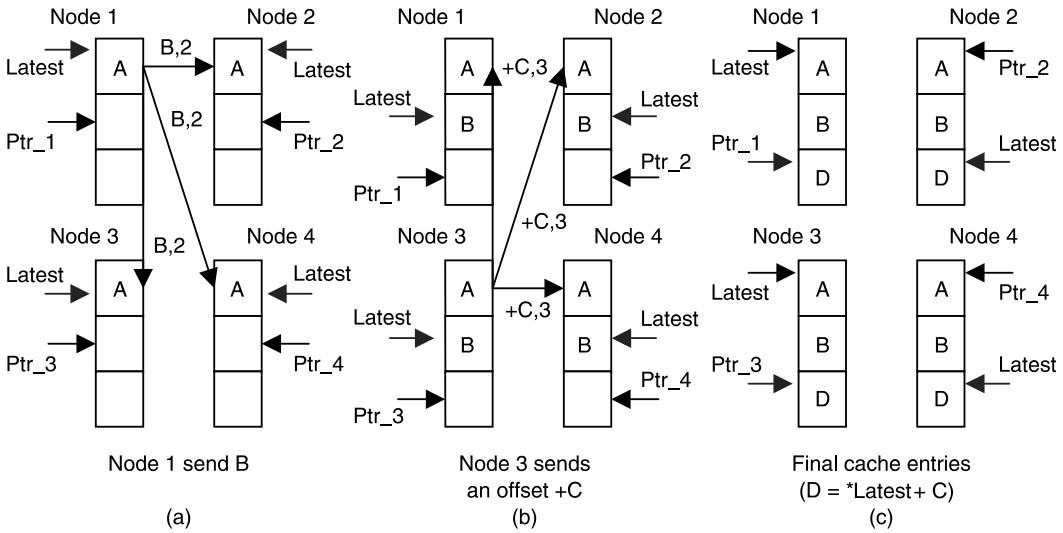


Figure 35.14. The situation when a node sends a value-difference packet.

We use the same configuration as the previous scheme discussed earlier in this experiment. First, we evaluated the comparison of energy savings of our spatial locality approach and the previous approach by varying the value locality rates, assuming a perfect wireless channel and that the threshold size is half of the data size, as shown in Figure 35.15.

From Figure 35.15, we see that the value-difference packet is useful when we have a low temporal locality rate. When we have only a 10% temporal locality rate and a 20% spatial locality rate, the proposed approach has the same amount of energy consumption as the original approach. However, if we have a 10% temporal locality rate and a 90% spatial locality rate, then energy is reduced by 10% compared with the original scheme. The original scheme is the method that does not apply any of our proposed schemes.

Figure 35.16 shows the comparisons between these two approaches by varying the lost rate, and Figure 35.17 shows the effect of varying the threshold size. Based on Figure 35.17, we have more reduction in the energy consumption when we reduce the size of the threshold assuming that the spatial

© 2005 by Chapman & Hall/CRC

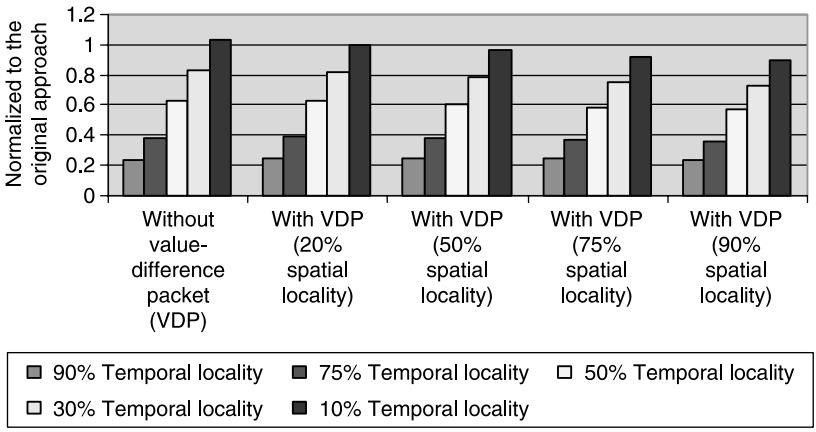


Figure 35.15. Energy consumption comparison between using value-difference packet and without using value-difference packet by varying locality rates. The original approach is the one that does not apply any of our proposed schemes.

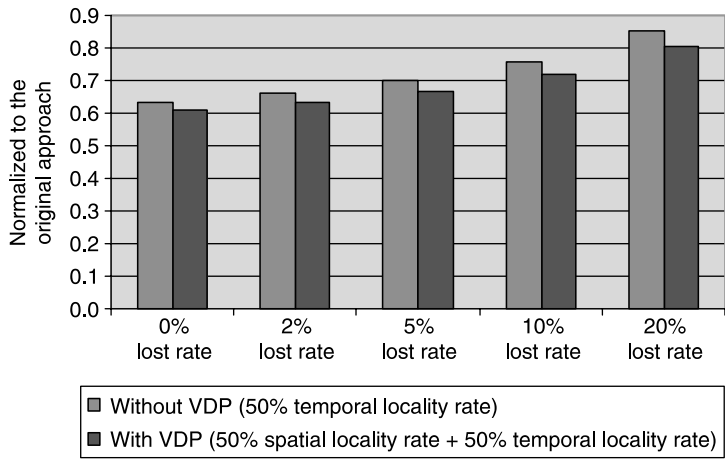


Figure 35.16. Energy consumption comparison between using value-difference packet (VDP) and without using value-difference packet by varying lost rates.

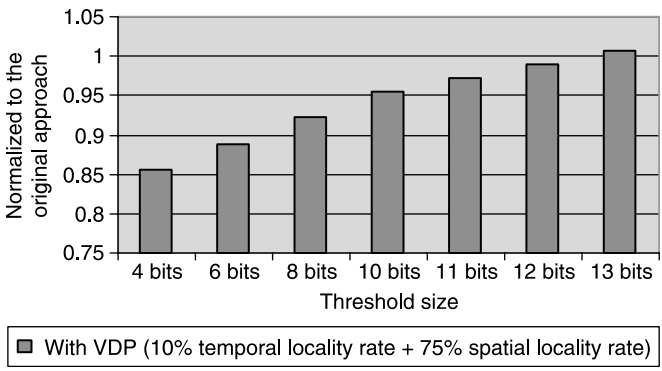


Figure 35.17. Energy consumption comparison between using value-difference packet (VDP) and without using value-difference packet by varying lost rates.

© 2005 by Chapman & Hall/CRC

locality rate remains the same. From these three figures (Figures 35.15–35.17), we can conclude that we can achieve more energy savings than without exploiting the possibility of spatial value locality.

35.6 Related Work

Several studies have focused on optimizing routing protocol, optimizing energy consumption, and improving security protocol for distributed sensor networks [6,14–19, 26].

Shah and Rabaey [15] present energy-aware routing for low-energy sensor networks by considering network survivability as their primary metric. They show that the network lifetimes can be increased up to 40% over comparable schemes like directed diffusion routing. Nath and Niculescu [17] show that simple trajectories can be used in implementing important network protocols, such as flooding, discovery, and network management. This technique will reduce the number of transmissions needed for such network protocols. This reduction will also give lower energy consumption. The work of Ghiasi *et al.* [16] focuses on how we can cluster the sensor nodes such that the energy consumption can be optimized. Our work is different, in that it specifically targets value locality due to spatial or temporal correlations for optimizing the communication across nodes in a cluster.

The study Yuan and Qu [18] shows how we can reduce the energy consumption of the encryption algorithm used in sensor networks using dynamic voltage scaling. Because sensor networks usually have a lot of small sensors, it is better to use very limited resources in each node in order to reduce the cost. However, when we use a system with limited capability [6] we have to consider using cryptography algorithms that require small resources, such as memory space. Perrig *et al.* [6] show that using the RSA algorithm on an Atmel processor (8-bit processor) is not possible due to the large resources requirements of the RSA algorithm. They present another security protocol that is suitable for sensor networks that use low-capability processors, called SPINS. This protocol uses the modified RC5 cryptography algorithm. Our work is complementary to these efforts, in that it attempts to eliminate some of the cryptographic operations by coding the transmitted data using the indexes in the cache.

35.7 Conclusions

In this chapter we have presented a new communication protocol for sensor nodes to reduce the overall energy consumption. The proposed technique also provides an inherent security to the transmitted data, thereby reducing the cost associated with cryptographic techniques. The energy savings of the proposed approach come from exploiting the value locality that a sensor network exhibits when communicating data (transmitting) within its local cluster and from reducing the number of encryption processes required. The evaluation of our technique shows that it is effective in reducing energy under different application characteristics, such as message intervals and value locality, as well as wireless channel conditions.

Acknowledgments

This work was supported in part by NSF CAREER 0093085 and 0093082 and NSF Award 0082064, 0103583, and 0202007.

References

[1] Ainsworth, D., Smart sensor technologies promise big savings in state energy costs, *Barkeleyan*, May, 2001.

[2] Eng, P., Tiny fire marshals. Dust-sized sensors could provide early warnings of forest fires, *ABC News*.

[3] Hollar, S., COTS dust, Master's Thesis, UC Berkeley, 2000.

[4] Estrin, D., Comm 'n Sense: research challenges in embedded networked sensing, Presentation at UCLA Computer Science Department Research Review, April 27, 2001.

© 2005 by Chapman & Hall/CRC

[5] Brooks, R.R., Reactive sensor network, Applied Research Laboratory, Pennsylvania State University.
[6] Perrig, A. *et al.*, SPINS: security protocols for sensor networks, in *Proceedings of MOBICOM*, 2001.
[7] Heinzelman, W.R. *et al.*, Energy-scalable algorithms and protocols for wireless microsensor networks, in *Proceedings of International Conference on Acoustics, Speech, and Signal Processing (ICASSP '00)*, June 2000.
[8] Hodjat, A. and Verbauwhede, I., AES module C code written using the suggested NIST C, University of California, Los Angeles, CA.
[9] Daemen, J. and Rijmen, V., The block cipher Rijndael, in *Smart Card Research and Applications, LNCS 1820*, Quisquater, J.-J. and Schneier, B. (eds), Springer-Verlag, 2000, 288.
[10] Daemen, J. and Rijmen, V., Rijndael, the advanced encryption standard, *Dr. Dobb's Journal*, 26(3), 137, 2001.
[11] Kaplan, I., RC5 source code, http://www.bearcave.com/cae/chdl/rc5.html (last accessed on January 2002).
[12] Schneier, B., *Applied Cryptography*, 2nd ed., John Wiley, 1996.
[13] Sinha, A. and Chandrakasan, A.P., JouleTrack — a Web based tool for software energy profiling, in *38th Design Automation Conference*, June 18–22, 2001.
[14] Degalahal, V. *et al.*, Analyzing soft errors in leakage optimized SRAM designs, in *Proceedings of International Conference on VLSI Design*, January 2003.
[15] Shah, R C. and Rabaey, J., Energy aware routing for low energy *ad hoc* sensor networks, in *IEEE Wireless Communications and Networking Conference (WCNC)*, Orlando, FL, March 17–21, 2002.
[16] Ghiasi, S. *et al.*, Optimal energy aware clustering in sensor network. *Sensor* 2, 258, 2002.
[17] Nath, B. and Niculescu, D., Routing on a curve, in *First Workshop on Hot Topics in Network (HotNets-I)*, Princeton, NJ, October 28–29, 2002.
[18] Yuan, L. and Qu, G., Design space exploration for energy-efficient secure sensor network, in *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP'02)*, 2002.
[19] Raghunathan, V. *et al.*, Energy aware wireless sensor networks, *IEEE Signal Processing Magazine*, 19(2), 40, 2002.
[20] Min, R. *et al.*, An architecture for a power-aware distributed microsensor node, *2000 IEEE Workshop on Signal Processing Systems (SiPS '00)*, October 2000.
[21] Hubaux, J.H.-P. *et al.*, The quest for security in mobile *ad hoc* network, *ACM Symposium on Mobile Ad Hoc Networking and Computing*, 2001.
[22] Kim, H.S. *et al.*, Multiple access caches: energy implications, in *Proceedings of the IEE CS Annual Workshop on VLSI*, Orlando, FL, April 27–28, 200, 53.
[23] Hodjat, A. and Verbauwhede, I., Power measurements and energy efficient implementations of network security algorithms for wireless sensor networks, in *Annual Research Review 2001, Electrical Engineering Department, UCLA.*
[24] Feng, J. *et al.*, System architecture for sensor network issues, alternatives, and directions, in *2002 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, 2002.
[25] Min, R. *et al.*, Dynamic voltage scaling techniques for distributed microsensor networks, in *Workshop on VLSI (WVLSI '00)*, April 2000.
[26] Chandrakasan, A. *et al.*, Power aware wireless microsensor systems, *Keynote Paper ESSCIRC,* Florence, Italy, September 2002.

© 2005 by Chapman & Hall/CRC

36

Compiler-Directed Communication Energy Optimizations for Microsensor Networks

I. Kadayif, M. Kandemir, A. Choudhary, M. Karakoy, N. Vijaykrishnan, and M.J. Irwin

36.1 Introduction and Motivation

A networked system of inexpensive and plentiful microsensors, with multiple sensor types, low-power embedded processors, and wireless communication and positioning ability offers promising solutions for many military and civil applications. As noted by Hill *et al.* [1], technological progresses in integrated, low-power CMOS communication devices and sensors make a rich design space of networked sensors viable. Recent years have witnessed several efforts at the architectural and circuit level for designing and implementing microsensor-based networks (e.g. see [2–6] and the references therein). While architectural/circuit-level techniques are extremely critical for the success of these networks, software optimizations are also expected to become instrumental in extracting the maximum benefits from the performance and energy behavior angles. In particular, large-scale data management techniques (at the computation and communication levels) are very important [7].

Optimizing energy consumption of a wireless microsensor network is important not only because it is not possible (in some environments) to recharge batteries on nodes (due to environment-related issues), but also because software running on sensor nodes may consume a significant amount of energy. In broad terms, we can divide the energy expended during operation into two parts: computation energy and communication energy. To minimize the overall energy consumption, we need to minimize the energy spent in both computation and communication. While power-efficient customized wireless protocols, e.g. [8,9], are part of the big picture as far as minimizing communication energy is concerned, we can also employ application-level and compiler-level optimizations that target at reducing communication energy.

© 2005 by Chapman & Hall/CRC

In this chapter, we focus on a wireless microsensor network environment that processes array-intensive codes. Note that array-intensive codes are very common in many embedded image- and signal-processing applications [10]. Focusing on a sensor network where the nodes form a two-dimensional mesh, we present a set of source-code-level communication optimization techniques and evaluate them from the energy perspective. Such networks can typically be found in several application domains, such as vehicle tracking, CAD-based imaging, and building/road protection. While our communication optimization techniques can be applied by programmers, the existing optimizing compiler technology can also be used to automate them. Specifically, in this chapter, we make the following contributions:

- We present the energy behavior of a set of array-intensive applications on a wireless microsensor network. Our results indicate that, for some applications, the computation energy dominates while for some others the communication energy dominates. More importantly, the communication energy profile is strongly dependent on how the arrays (datasets) are decomposed (distributed) across memories of sensor nodes.
- We explain how source-level communication optimizations can reduce the energy consumption during communication, and present experimental data. We show that, in some cases, optimizing communication energy aggressively can even shift the energy bottleneck from communication to computation.
- We report on the results of our sensitivity analysis where we modify several parameters in our energy models. The objective of this analysis is to observe how these changes affect the energy benefits coming from the compiler optimizations.
- We present a strategy where we overlap the communication time with the computation time. Our experimental results indicate that such an overlap can reduce the leakage energy consumption of sensor nodes significantly. Note that leakage energy is expected to play a major role in upcoming process technologies [11].
- Based on the experimental data collected, we present a compiler algorithm that applies several communication optimizations in a unified framework for optimizing the energy spent in a sensor network during execution.

This chapter is a first step in using high-level (source-code-level) compiler optimizations for reducing energy consumption in sensor networks. Our energy savings show that software-level optimizations can be very useful in prolonging the lifetime of these networks. Our communication optimizations are also, in a sense, complementary to the traditional signal mapping strategies in the literature, e.g. [12,13].

The remainder of this chapter is organized as follows. Section 36.2 gives a high-level view of the architecture assumed in this study. Section 36.3 discusses the communication optimizations considered in this work and explains how they help reduce energy consumption. Section 36.4 presents the computation and communication energy models used in our work. Section 36.5 gives experimental data that show the effectiveness of our approach. Section 36.6 puts our major observations into perspective and presents a unified compiler algorithm. Finally, Section 36.7 concludes the chapter with a summary of our major contributions and a brief discussion of future work on this topic.

36.2 High-Level Architecture

The upper part of Figure 36.1 shows the sensor network architecture assumed in this work. Basically, we assume that the sensor nodes are distributed over a two-dimensional space (area), and the distance between neighboring sensors is close to uniform throughout the network. In our study, each sensor node is assumed to have the capability of communicating with its four neighbors. Each node in this network can be identified using coordinates `a` (in the horizontal dimension) and `b` (in the vertical dimension), and can be denoted using `P(a,b)`. Consequently, its four neighbors can be identified using `P(a-1,b)`, `P(a+1,b)`, `P(a,b-1)`, and `P(a,b+1)`. Note that the nearest-neighbor

© 2005 by Chapman & Hall/CRC

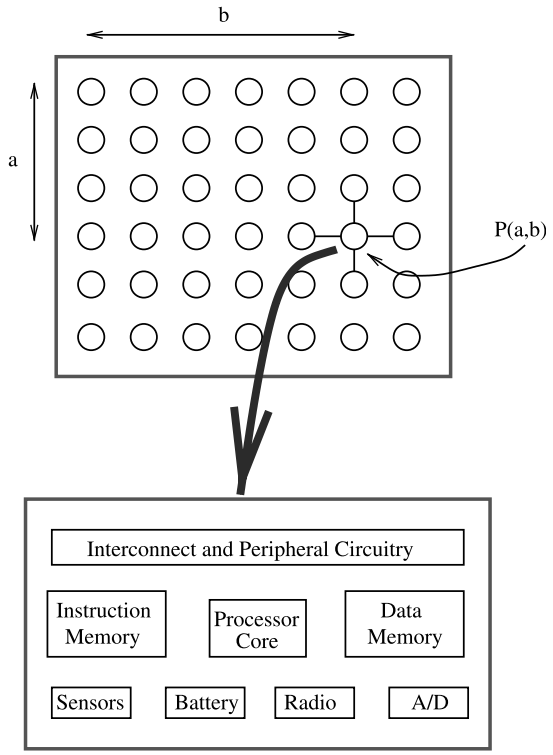


Figure 36.1. Assumed sensor network architecture and blocks in a sensor node.

communication style matches very well with many real-world uses of microsensors, as the neighboring nodes are expected to share data to carry out a given task [14]. It should also be noted, however, that, although not discussed here, our software framework is able to handle general node-to-node or broadcast types of communication as well.

The lower part of Figure 36.1 illustrates the major components of a given sensor node. Each node in our network contains sensor(s), A/D converter, battery, processor core, instruction and data memories, radio, and peripheral circuitry. Since sensor networks can be deployed in time-critical applications, we do not consider a cache architecture. Instead, each node is equipped with fast (SRAM) instruction and data memories. The software support for such an architecture is critical. Our compiler takes an input code written in C and *parallelizes* it across the nodes in the network. After parallelization, each sensor node executes the same code (parameterized using parameters `a` and `b`) but works on a different portion of the dataset (e.g. a rectilinear segment of a multidimensional array of signals). In other words, the sensor nodes effectively exploit data parallelism. As explained below in detail, the compiler parallelizes each loop nest in the code using the data decomposition supplied by the programmer. For a given array, a data decomposition specifies how the elements of the array are decomposed (distributed) across the memories of sensor nodes. When an array element is mapped to the memory of a sensor node, that sensor node is said to *own* the array element.

This style of parallel sensor operation can be found in diverse application domains from vehicle tracking to earthquake studies. For example, in the vehicle tracking/detection domain, the sensors can be placed regularly to form a two-dimensional mesh structure in the area that needs to be protected [15]. A parallel application (vehicle detection software) continuously runs and checks to see whether there is a vehicle in the area, and if so the sensors collaboratively track it. During this tracking activity, sensors frequently engage in communication to share data for accurate tracking (and for vehicle identification). Obviously, minimizing inter-node communication can help reduce the overall communication energy dramatically.

© 2005 by Chapman & Hall/CRC

36.2.1 Language Support

In order to express communication at the source language level, we assume that two communication primitives are available. The first primitive is of the form

```
send DATA to P(c,d)
```

When executed by a node `P(a,b)`, this primitive sends data (denoted by `DATA`) to processor `P(c,d)`. The data communicated can be a single array element, an entire array, or (in many cases) an array region. We assume that this data is received by the node `P(c,d)` when it executes our other primitive:

```
receive DATA from P(a,b)
```

In order for a communication to occur, each send primitive should be matched with a corresponding receive primitive. Note also that all communication protocol-related activities are assumed to be captured within these primitives. If the size of the message indicated in these calls is larger than the packet size, then the message is divided into several packets. This activity occurs within the `send` and `receive` routines. While the efficient implementation of these high-level primitives is an important topic in itself, it is beyond the scope of this chapter. This chapter, rather, deals with the problem of how these primitives can be used by an optimizing compiler. Our compiler framework takes an original code and after generating the node program (i.e. the program that will be run on the sensor nodes) inserts these send/receive primitives automatically (without user involvement).

36.3 Communication Optimizations

36.3.1 Data Decomposition and Parallelization

We focus on applications where arrays of signals are processed by multiple sensor nodes in parallel. In such applications, given an array of signals, typically, each processor is responsible from processing a portion of the array. Note that this operation style matches directly to an environment where each sensor node is collecting some data from the portion of an area covered by it and processing the collected data.

Our parallelization strategy works on a single nest at a time; that is, each nest in the application code being optimized is parallelized independently of the other nests. In order to parallelize a given nest over sensor nodes, we need to perform two tasks: (i) decomposing arrays of signals across the memories of sensor nodes and (ii) distributing loop iterations across nodes. Note that array decomposition and loop iteration distribution, together, achieve parallelism across sensor nodes. In this chapter, a decomposition for an m-dimensional array of signals is specified by application programmer using a notation of the form `[D1][D2]...[Dm]` at the beginning of application, where each `Di` can be either an asterisk, meaning that the corresponding dimension is not decomposed, or `block`, meaning that one block of adjacent elements is assigned to each node memory. For example, Figure 36.2(a) shows how an array is decomposed across eight sensor nodes. Note that each sensor node takes a row–block of the array. This decomposition is expressed as `[block][*]`, indicating that the first dimension is decomposed across the sensor nodes whereas the second one is not. Figure 36.2(b), on the other hand, shows the `[*][block]` decomposition on eight sensor nodes. Finally, Figure 36.2(c) shows how an array is decomposed in both the dimensions (i.e. a `[block][block]` decomposition) using 16 sensor nodes.

In this work, we adopt an array decomposition oriented parallelization strategy based on the owner-computes rule used by optimizing compilers [16]. In this strategy, an array element is updated (written) by only the node that owns it. Let us assume that $\{I\}$ is the set of iterations that will be executed by a given loop nest and that $\{D\}$ is the set of data elements (array elements) that will be used in the computation within the loop. Assume further that $\text{dist}(\{D\}) \longrightarrow (p, \{D_p\})$ is a decomposition function that gives the set of data elements $\{D_p\}$ mapped to sensor node p. Let r_0, r_1, $r_2, \ldots, r_s$ be $(s+1)$ references in an assignment statement in the loop where r_0 is the left-hand-side (LHS) reference and the

© 2005 by Chapman & Hall/CRC

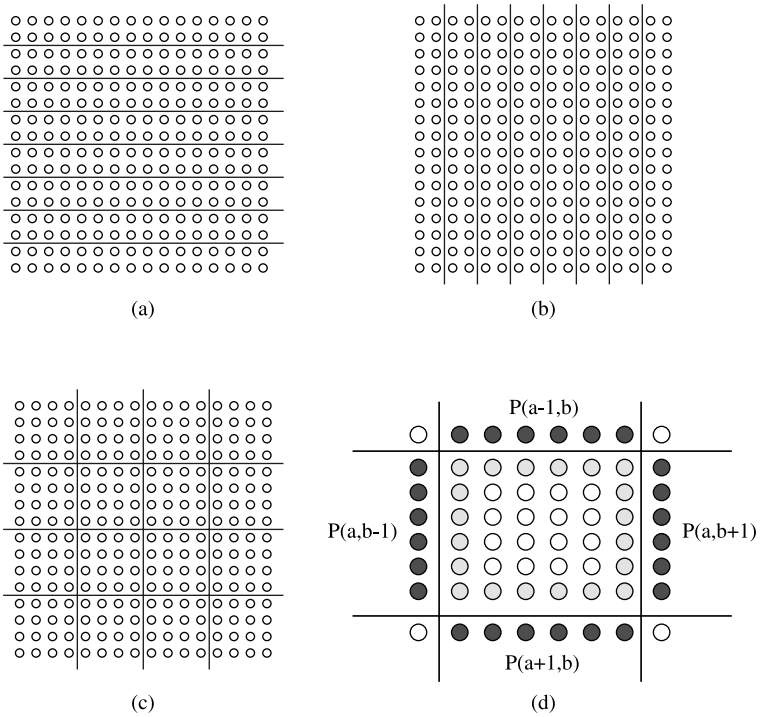


Figure 36.2. (a)–(c) Decomposition of a two-dimensional array of signals across sensor nodes. (d) Communication requirements of a sensor node `P(a,b)`.

remaining ones are right-hand-side (RHS) references. For a given reference r_k, we define a function $\text{subs}_{r_k}(.)$ as a mapping from $\{I\}$ to the set of array elements accessed by this reference, $\{D'_k\}$. That is, $\{D'_k\}$ is the set of array elements accessed through r_k when the iterations in $\{I\}$ are executed. Then, the local index set $\{I_p\}$ for node p can be defined as

$$\{I_p\} = \{i | i \in \text{subs}_{r_k}^{-1}(\{D'_0\} \cap \{D_p\})\}$$

Note that $\{I_p\}$ represents the set of iterations that assign values to the elements of p accessed by the LHS reference r_0. That is, the iterations in $\{I_p\}$ are the ones that will be executed by sensor node p (according to the owner-computes rule), as these are the iterations that assign values to the array elements owned by p. It should also be noted that these iterations also access data elements using RHS references. We can express the set of these elements as

$$\{D_{\text{RHS}_p}\} = \bigcup_{k=1}^{s} \text{subs}_{r_k}(\{I_p\}).$$

Consequently, the elements in set $\{D_{\text{RHS}_p}\} - \{D_p\}$ are the elements that sensor node p needs to receive from other processors. This is called the receive-set. The send-set of a given sensor node can also be computed in a similar manner. To summarize, our approach takes data (array) decompositions into account and, using the array references in the code and information about loop iterations, computes the elements that each processor needs to receive and send. Note that, unless its $\{D_{\text{RHS}_p}\} - \{D_p\}$ set is empty, a sensor node needs communication before completing its part of the workload [17].

36.3.2 Naive Communication

As explained above, during parallel execution, processors may need to engage in communication with each other. This is because in manipulating the array elements in its portion a node can require array

© 2005 by Chapman & Hall/CRC

elements that belong to the portions of arrays owned by some other nodes. Consider, for example, the following application code fragment which is to be parallelized across 256 sensor nodes (for illustrative purposes) that form a 16×16 grid:

```
for(i=2;i<=95;i++)
  for(j=2;j<=95;j++)
    u[i][j] = (v[i-1][j]+v[i+1][j]
              +v[i][j-1]+v[i][j+1])/4;
```

Assuming a `[block][block]` data decomposition for both 96×96 arrays `u` and `v`, a (nonboundary) node needs data from each of its four neighbors to compute new values for some of the elements in its portion of array `u`. This scenario is illustrated in Figure 36.2(d) for a node `P(a,b)`. As will be demonstrated shortly, in general there might be different ways of implementing the communications required by this fragment. In this subsection we describe the most straightforward (naive) strategy; the following subsections focus on communication optimizations with different levels of sophistication.

In the naive communication model, each sensor node executes a code fragment similar to the one shown below. Note that, in addition to receiving elements from its neighbors, a node also sends elements to its neighbors. The following fragment is the code that will be executed by each node `P(a,b)`. Figure 36.2(d) shows the set of elements that need to be received by node `P(a,b)` from its neighbors (shown as shaded dark) and the set of elements that will be sent by node `P(a,b)` to its neighbors (shown as lightly shaded). In the code below, within the loops, node `P(a,b)` first sends the elements that will be needed by its neighbors, and then receives the elements that it needs from its neighbors.

```
for(i=1;i<=6;i++)
  for(j=1;j<=6;j++)
    {
     if (i==1) send v[i][j] to P(a-1,b);
     if (i==6) send v[i][j] to P(a+1,b);
     if (j==1) send v[i][j] to P(a,b-1);
     if (j==6) send v[i][j] to P(a,b+1);
     if (i==1) receive v[i-1][j] from P(a-1,b);
     if (i==6) receive v[i+1][j] from P(a+1,b);
     if (j==1) receive v[i][j-1] from P(a,b-1);
     if (j==6) receive v[i][j+1] from P(a,b+1);
       u[i][j] = (v[i-1][j]+v[i+1][j]
                 +v[i][j-1]+v[i][j+1])/4;
    }
```

It should be noted that the original array sizes (for both `u` and `v`), 96×96, are decomposed across (16×16) sensor nodes and each node (after the decomposition) owns a 6×6 section of the arrays. Note also that, after applying the owners-compute rule, each node executes its local index set, which consists of a total of 6×6 loop iterations. It should be stressed that, in general, it may not be possible to perform all communications (or to replicate some array elements) before the computation begins. This is because data dependences in the code can require that the new value of a data item needs to be computed first (by the owner node) before it can be communicated to a neighboring node.

36.3.3 Message Vectorization

Placing communication calls (the send/receive statements) within the innermost loop can increase the energy spent in communication dramatically. This is due to three main reasons. First, each such

© 2005 by Chapman & Hall/CRC

communication placement invokes a new communication for each iteration of the enclosing loops. Second, each such communication typically sends/receives a small number of array elements. For example, in the naive communication case above, each communication call sends (or receives) only a single array element. Assuming 2 bits for each array element (e.g. for an image that uses four colors), it is easy to see that naive communication causes a large communication overhead. Third, the if-statement used in the inner loop positions can degrade the performance of the processor core in a sensor node dramatically, thereby incurring an energy overhead. Consequently, an optimization that extracts communication from within loops and combines element messages per loop iteration in one vectorized message preceding the loop can be very useful. Such an optimization is termed message vectorization [18,19] and can be applied based on the results of data dependence analysis [16]. More specifically, given a loop nest, this optimization is performed in two steps. In the first step, the nest is analyzed and the outermost loop level above which the communication can be performed is determined. Note that this is the loop level where element messages resulting from the same array reference may legally be combined (into a vectorized message). In the second step, the communication calls (send/receive statements) are inserted in the code. The following example shows the message vectorized version of our example above:

```
send v[1][1..6] to P(a-1,b);
send v[6][1..6] to P(a+1,b);
send v[1..6][1] to P(a,b-1);
send v[1..6][6] to P(a,b+1);
receive v[0][1..6] from P(a-1,b);
receive v[7][1..6] from P(a+1,b);
receive v[1..6][0] from P(a,b-1);
receive v[1..6][7] from P(a,b+1);
for(i=1;i<=6;i++)
  for(j=1;j<=6;j++)
    u[i][j] = (v[i-1][j]+v[i+1][j]
              +v[i][j-1]+v[i][j+1])/4;
```

Note that, in this message-vectorized fragment, the entire communication is hoisted above the nest. Note also that each send/receive call communicates six array elements.[1] While this optimization does not reduce the number of elements sent/received, it significantly reduces the number of times the communication will be performed. Since in sensor networks each new communication initiation has an energy cost, this optimization can reduce the communication energy costs significantly.

36.3.4 Message Coalescing

This is an optimization that targets at eliminating the communication of redundant data from one sensor node to another. If the sets of array elements that will be communicated due to two different references to the same array overlap (i.e. contain common elements), then message coalescing transfers these elements only once [16]. It is typically applied after message vectorization. For example, consider the following program fragment:

```
for(i=2;i<=95;i++)
  for(j=2;j<=95;j++)
    u[i][j] = (v[i-1][j]+v[i-2][j]
              +v[i][j-1]+v[i][j-2])/4;
```

[1]A notation such as $v[6][1 \ldots 6]$ means all elements in the set $\{v[6][1], v[6][2], v[6][3], v[6][4], v[6][5], v[6][6]\}$.

© 2005 by Chapman & Hall/CRC

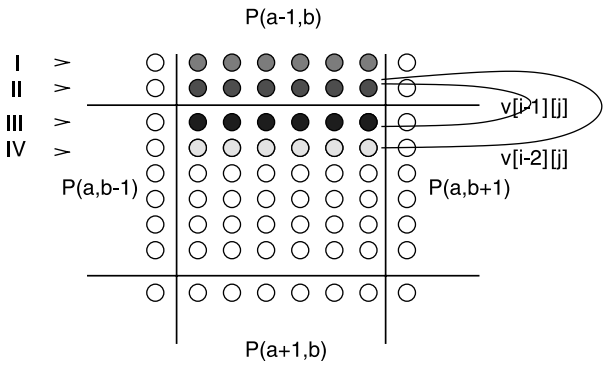


Figure 36.3. Overlapping receive sets.

Note that in this code fragment the communications due to `v[i-1][j]` and `v[i-2][j]` overlap. That is, the receive-sets due to these two references contain some common array elements. If each receive-set is vectorized independently, then the same array element would be transferred twice. To illustrate this, let us consider Figure 36.3, which shows the communications due to references `v[i-1][j]` and `v[i-2][j]`. Note that to compute the new values of row III of its portion of array `u`, node `P(a,b)` needs the rows I and II (of array `v`) from its neighbor `P(a-1,b)`. Similarly, to compute the new values of row IV (of array `u`), it needs the row II (of array `v`) from `P(a-1,b)`. In other words, node `P(a,b)` requires the same row (i.e. row II) from `P(a-1,b)` twice. Instead of performing a separate communication for each request, message coalescing combines these communications into a single (vectorized) message. The same scenario also occurs with references `v[i][j-1]` and `v[i][j-2]` (when communicating with sensor node `P(a,b-1)`).

36.3.5 Message Aggregation

The two optimizations discussed so far try to reduce communication due to a single array of signals. That is, they are applied to each array independently. Message aggregation, in contrast, targets multiple arrays and tries to ensure that only one message is sent (from a given sensor node) to each sensor node. It is usually applied after message vectorization and message coalescing, and combines all data that will go to the same node into a single message. Note that, to implement this optimization, an extra level of buffering might be required. More specifically, during code generation, the array elements to be aggregated are copied to a single buffer so that they can be sent as a single message. The receiving processor then copies the buffered data back to the appropriate locations in its memory. The following code fragment illustrates a case where message aggregation can be applied. In this fragment, there are communications due to two different arrays (`v` and `w`). Message aggregation combines these communications into one communication.

```
for(i=2;i<=95;i++)
  for(j=2;j<=95;j++)
    u[i][j] = (v[i-1][j]+w[i-1][j]
              +v[i][j+1]+w[i][j+1])/4;
```

It should be noted that these three communication optimizations, namely message vectorization, coalescing, and aggregation, do not have too much impact on computation energy. Although these optimizations reduce the number of send/receive calls inserted in the code, their overall impact on computation energy is not expected to be significant. In fact, our experiments revealed that the maximum computation energy variance due to these optimizations was 1.2%.

© 2005 by Chapman & Hall/CRC

36.3.6 Inter-Nest Optimization

A common characteristic of all the three communication optimization discussed in previous subsections is that they work on a single nest at a time. While this might make the user's or compiler's job easier, in some cases it may also lead to unnecessary communication (and extra energy consumption). One such scenario occurs, for example, when the sets of communications required by two successively executed nests overlap. To illustrate this, we consider the following code fragment:

```
for(i=2;i<=95;i++)
  for(j=2;j<=95;j++)
    u[i][j] = (v[i-1][j]+v[i+1][j]
              +v[i][j-1]+v[i][j+1])/4;
for(i=2;i<=95;i++)
  for(j=2;j<=95;j++)
    w[i][j] = (v[i-1][j]+v[i+1][j]
              +v[i][j-1]+v[i][j+1])/4;
```

In this code fragment, assuming the same data decomposition for all the arrays involved, the communication requirements for a given sensor node `P(a,b)` are the same in both the nests. That is, in both the nests the same processor needs to receive the same elements of array `v`. While message vectorization can optimize each nest individually, the communication due to the second nest would lead to a wasted energy consumption. An inter-nest optimization captures this redundant communication and optimizes it away. In this example, what this optimization does is just to eliminate the communication before the second nest. In general, however, applying this optimization can be more difficult. For example, if some elements of array `v` are updated between these two nests (by their owner nodes), these updated elements still need to be transferred (e.g. before the second nest above). In mathematical terms, if (for a given sensor node) D' is the receive set due to the first nest, D'' is the receive set due to the second nest, and D_u is the set of updated elements (i.e. between these two nests), the elements in $D' \cup (D'' - D_u)$ can be transferred before the first nest and the elements in D_u can be transferred before the second one. Here, $\cup$ and $-$ refer to set union and set subtraction respectively. While not all applications benefit from this optimization, in cases where it is applicable the energy benefits might be very significant.

However, it should be mentioned that, unlike previous optimizations that target a single nest at a time, this optimization in general increases the computation energy consumed. This is because, as explained above, this optimization performs set arithmetic on array regions that come from different nests. These operations, in general, demand construction of new loops to collect the elements to be operated on, and are costly from both execution time and energy perspectives. In fact, as will be shown later in the chapter, in some cases the increase in the computation energy can offset potential energy gains coming from the optimized communication. Therefore, inter-nest optimizations should be applied with care.

36.4 Experimental Setup

36.4.1 Benchmark Codes

We use a set of array-intensive benchmark programs in our experiments. The salient characteristics of the codes in our experimental suite are summarized in Table 36.1. The first, third, fourth, and sixth benchmarks are motion estimation codes. The second one is an alternate direction integral code. `mxm` and `tomcatv` are an integer matrix multiplication code and a mesh generation code respectively. The last two codes, Jacobi relaxation and red-black successive over-relaxation (SOR), contain stencil-like computations and reductions, the two techniques commonly used in image and video processing. Each array element is assumed to be 4 bits wide.

© 2005 by Chapman & Hall/CRC

Table 36.1. Benchmarks used in the experiments. The second and third columns give, respectively, the total input size and the number of arrays in the code

Benchmark	Input size (KB)	Number of arrays	Brief description
`3-step-log`	295.08	3	Motion estimation
`adi`	271.09	6	Alternate direction integral
`full-search`	98.77	3	Motion estimation
`hier`	97.77	7	Motion estimation
`mxm`	464.84	3	Matrix multiply
`parallel-hier`	295.08	3	Motion estimation
`tomcatv`	174.22	9	Mesh generation
`jacobi`	312.00	2	Stencil-like computation
`red-black SOR`	156.00	1	Stencil-like computation

36.4.2 Modeling Energy Consumption

Dynamic energy consumption due to switching of hardware components is dependent strongly on how different components of a sensor node are exercised by a given application [20]. We separate the system energy into two parts: computation energy and communication energy. Computation energy is the energy consumed in the processor core (datapath), instruction memory, data memory, and clock network. In this work, we focus on a simple, single-issue, five-stage pipelined processor core which is suitable for employing in a sensor node. This core has instruction fetch (IF), instruction decode/operand fetch (ID), execution/address calculation (EXE), memory access (MEM), and write-back (WB) stages. We use SimplePower [21], a publicly available, cycle-accurate energy simulator, to model the energy consumption in this processor core. The modeling approach used in SimplePower has been validated to be accurate (with an average error rate of 8.98%) using actual current measurements of a commercial DSP architecture [22].

We assume that each node has an instruction memory and a data memory (both are SRAM). The energy consumed in these memories is dependent primarily on the number of accesses and memory configuration (e.g. capacity, the number of read/write ports, and whether it is banked or not). We modified the Shade simulation environment [23] to capture the number of references to instruction and data memories and used the CACTI tool [24] to calculate the per access energy cost. The data collected from Shade and CACTI are then combined to compute the overall energy consumption due to memory accesses.

The clock generation circuit phase-locked loop [PLL], the clock distribution buffers and wires, and the clock-load on the clock network presented by the clocked components are the main energy consumers for the clock network in our sensor node. We enhanced SimplePower to estimate the clock network energy consumption in each cycle by determining which parts of the clock network are active and using the corresponding energy models for active components.

As our communication energy component, we consider the energy expended for sending/receiving data. The radio in the sensor nodes is capable of both sending data and, at the same time, sensing incoming data. We assume that if the radio is not sending any data, then it does not spend any energy (omitting the energy expended due to sensing). After packing data, the processor sends the data to the other processor via radio. The radio needs a specific startup time to start sending/receiving message. We used the radio energy model presented by Shih *et al.* [7] to account for communication energy. In this model, the power equation of the radio is expressed as

$$P_{\text{radio}} = N_{\text{tx}}[P_{\text{tx}}(T_{\text{on-tx}} + T_{\text{st}}) + P_{\text{out}}T_{\text{on-tx}}] + N_{\text{rx}}[P_{\text{rx}}(T_{\text{on-rx}} + T_{\text{st}})]$$

where $N_{\text{tx/rx}}$ is the average number of times per second that the transmitter/receiver is used; $P_{\text{tx/rx}}$ is the power consumption of transmitter/receiver; P_{out} is the output transmit power that drives the antenna; $T_{\text{on-tx/on-rx}}$ is the time interval required to send/receive data; and T_{st} is the startup time of the

© 2005 by Chapman & Hall/CRC

Table 36.2. Parameters used in our base configuration

Parameter	Value
$\|P\|$	160
Instruction memory	8 KB
Data memory	16 KB
P_{tx}	80 mW
P_{rx}	200 mW
T_{st}	450 μs
P_{out}	1 mW
L	250 bits
B	1 Mb/s

transceiver. Also, note that $T_{on-tx/on-rx} = L/B$, where L is packet size (message length in bits) and B is the data transmit/receive rate in bits per second.

Our *base configuration* uses the values given in Table 36.2. The power values in this table are similar to those used by Shih and co-workers [2,7]. In all our experiments we maintain that P_{rx} is equal to $2.5P_{tx}$ (as the receiver has more circuitry than the transmitter). That is, whenever P_{tx} is modified, P_{rx} is also modified accordingly. In Table 36.2, $|P|$ denotes the total number of sensor nodes that participate in parallel execution of the application.

36.5 Results

Our presentation is in four parts. In Section 36.5.1 we give an energy breakdown (between computation and communication) when different data decompositions and communication optimizations are used. In Section 36.5.2 we present a sensitivity analysis where we modify several parameters used in our base configuration. In Section 36.5.3 we evaluate the effectiveness of inter-nest communication optimization. Finally, in Section 36.5.4 we quantify the potential energy benefits of overlapping computation and communication.

36.5.1 Energy Breakdown

Table 36.3 gives the energy breakdown for our benchmarks for three different data decompositions and communication optimizations. Since the computation energies for different decompositions and different communication optimizations are almost the same, we report only one computation energy value for each benchmark. The column decomposition (in Table 36.3) refers to a decomposition whereby each sensor node owns a column–block of each array; the other decompositions correspond to cases where the arrays involved in the computation are decomposed in row–block and block–block manner across sensor nodes. Also, we consider three different communication optimizations: message vectorization (denoted `v`), message vectorization plus message coalescing (denoted `v+c`), and message vectorization plus message coalescing plus message aggregation (denoted `v+c+a`). Unless stated otherwise, all energy numbers given are for dynamic energy only (i.e. they do not include the leakage energy consumption).

We can make several observations from the numbers reported in Table 36.3. First, we see that in some benchmarks the computation energy dominates the communication energy, whereas in some benchmarks it is the opposite. This is largely a characteristic of the access pattern exhibited by the application under a given (data decomposition, communication optimization) pair. Based on these results, we can conclude that communication energy constitutes a significant portion of the overall energy budget. The second observation is that the communication optimizations save energy. For example, `v+c` improves the communication energy of the `hier` benchmark by 13.7% (over the only message-vectorized version) when column decomposition is used. As another example, the `v+c+a`

© 2005 by Chapman & Hall/CRC

Table 36.3. Computation and communication energy breakdown for our applications. All energy values are in microjoules

Benchmark		Communication energy			Computation energy				
		Column	Row	Block	Instr. Mem.	Data Mem.	Clock	Datapath	Total
3-step-log	v	6867	5922	12,064					
	v+c	5449	5449	5449	266,503	58,376	27,411	69,894	422,185
	v+c+a	3559	3559	3559					
adi	v	173,196	173,196	82,886					
	v+c	173,196	173,196	82,886	38,204	7734	8037	19,541	73,516
	v+c+a	165,636	165,636	36,894					
full-search	v	127,615	123,835	189,985					
	v+c	110,605	110,605	110,605	343,946	84,058	31,132	84,654	543,788
	v+c+a	108,715	108,715	108,715					
hier	v	191,422	185,752	284,977					
	v+c	165,907	165,907	165,907	24,961	5945	1902	5373	38,179
	v+c+a	164,017	164,017	164,017					
mxm	v	431,700	431,700	286,080					
	v+c	367,440	367,440	146,976	16,609	2653	2576	5734	27,572
	v+c+a	367,440	367,440	146,976					
parallel-hier	v	63,807	61,917	94,992					
	v+c	55,302	55,302	55,302	168,742	36,775	15,326	40,626	261,468
	v+c+a	55,302	55,302	55,302					
tomcatv	v	153,780	61,512	58,934					
	v+c	153,780	61,512	42,494	18,877	3788	3195	7844	33,706
	v+c+a	142,440	57,732	33,623					
jacobi	v	72,839	72,839	33,671					
	v+c	72,839	72,839	33,671	75,264	12,016	11,858	27,568	126,706
	v+c+a	72,839	72,839	33,671					
red-black-SOR	v	76,619	76,619	39,719					
	v+c	76,619	76,619	39,719	73,732	11,472	12,288	28,160	125,652
	v+c+a	76,619	76,619	39,719					

version in tomcatv saves 7.2% communication energy over the v+c version. In fact, as can be seen from our results, in some cases an optimization can even shift the energy bottleneck from communication to computation. For example, in the adi benchmark, when block decomposition is used, the communication energy of the v+c version is larger than the computation energy. However, when we use the v+c+a version, the communication energy becomes less than the computation energy. Third, we can observe that data decomposition in some benchmarks makes a difference in communication energy. For example, the block decomposition performs much better than column and row decompositions in adi. In fact, for this benchmark code, while communication energy dominates computation energy in column and row decompositions, communication energy is approximately half of the computation energy if the message optimizations are applied in conjunction with block decomposition.

It should be stressed that working with the naive communication strategy described earlier (i.e. not using any communication optimization) may result in an intolerable communication energy. To illustrate this, Table 36.4 shows the communication energy consumption when naive communication is employed. Comparing these values with those in Table 36.3 emphasizes the difference between optimizing and not optimizing communication energy. These results clearly show that communication optimizations are vital to keep the energy consumption of a sensor network under control.

© 2005 by Chapman & Hall/CRC

Table 36.4. Communication energy with naive communication. All energy values are in microjoules

Benchmark	Column	Row	Block
`3-step-log`	173,200,144	173,200,144	173,200,144
`adi`	2,429,856	2,429,856	971,942
`full-search`	360,833,632	360,833,632	360,833,632
`hier`	22,719,154	22,719,154	22,719,154
`mxm`	15,819,376	15,819,376	25,311,000
`parallel-hier`	28,866,690	28,866,690	28,866,690
`tomcatv`	2,024,880	809,952	570,206
`jacobi`	1,036,739	1,036,739	414,695
`red black SOR`	1,036,739	1,036,739	414,695

36.5.2 Sensitivity Analysis

To measure the sensitivity of communication optimizations to several parameters of the radio, we performed another set of experiments where only one parameter is modified at a time. The parameters modified are T_{st} (startup time), P_{tx} (transmitter power), and B (data rate). Note that some of these variations also help us (indirectly) evaluate the impact of different communication protocols. For example, increasing the number of error-control bits added by a protocol can be thought of as increasing P_{tx}. In this subsection, using the base configuration, we also experimented with different network sizes.

Figure 36.4 shows the effect of startup time on communication energy in the `adi` and `jacobi` benchmarks. From these graphs, we can make three observations. First, both the graphs show that the startup time has a larger impact with block decomposition than with row and column decomposition. This is due to the fact that, in block decomposition, the message lengths are generally smaller than those in column and row decompositions. Consequently, the startup time has a larger impact with block decomposition. The second observation is that a large startup time can bring the communication energy to the same order of magnitude as the computation energy. Our third observation emphasizes the importance of communication optimizations. One can see from the top graph in Figure 36.4 that the `v+c+a` version, when used in conjunction with the block decomposition, prevents the communication energy from significantly increasing when T_{st} is increased (compared with row and column decompositions). That is, message aggregation optimization can be vital in coping with the negative impact of large communication parameters. This is because what message aggregation does is to combine small messages from different arrays into a large message. In other words, it reduces the number of messages, which in turn makes the communication behavior less sensitive to the startup time.

Figure 36.5 illustrates the effect of transceiver power on communication energy in the `adi` and `full-search` benchmarks. As can be seen clearly from these figures, the communication energy increases almost linearly with the transceiver power P_{tx}. This is because the transceiver power is very large, compared with the transmit power of the antenna P_{out}, and it is the main factor that determines the overall trend in communication energy.

The effects of data transmit/receive rate on communication energy of the `adi` benchmark and the `jacobi` benchmark are shown in Figure 36.6. Since an increase in transmit/receive rate reduces transmit/receive time, the radio will need to be active for a smaller period of time to send/receive the message; consequently, the communication energy is reduced. Also note that, for very high rates, the startup time balances or dominates the transmit/receive time, so energy overhead due to startup time plays a very critical role in total communication time. As a consequence, the number of messages (rather than the total size of messages) determines the communication energy. While, in the `jacobi` benchmark, with a 1 MB/s transmit/receive rate, data communication energy for block decomposition is approximately half of the communication energy of column/row decomposition, at a 10 MB/s

© 2005 by Chapman & Hall/CRC

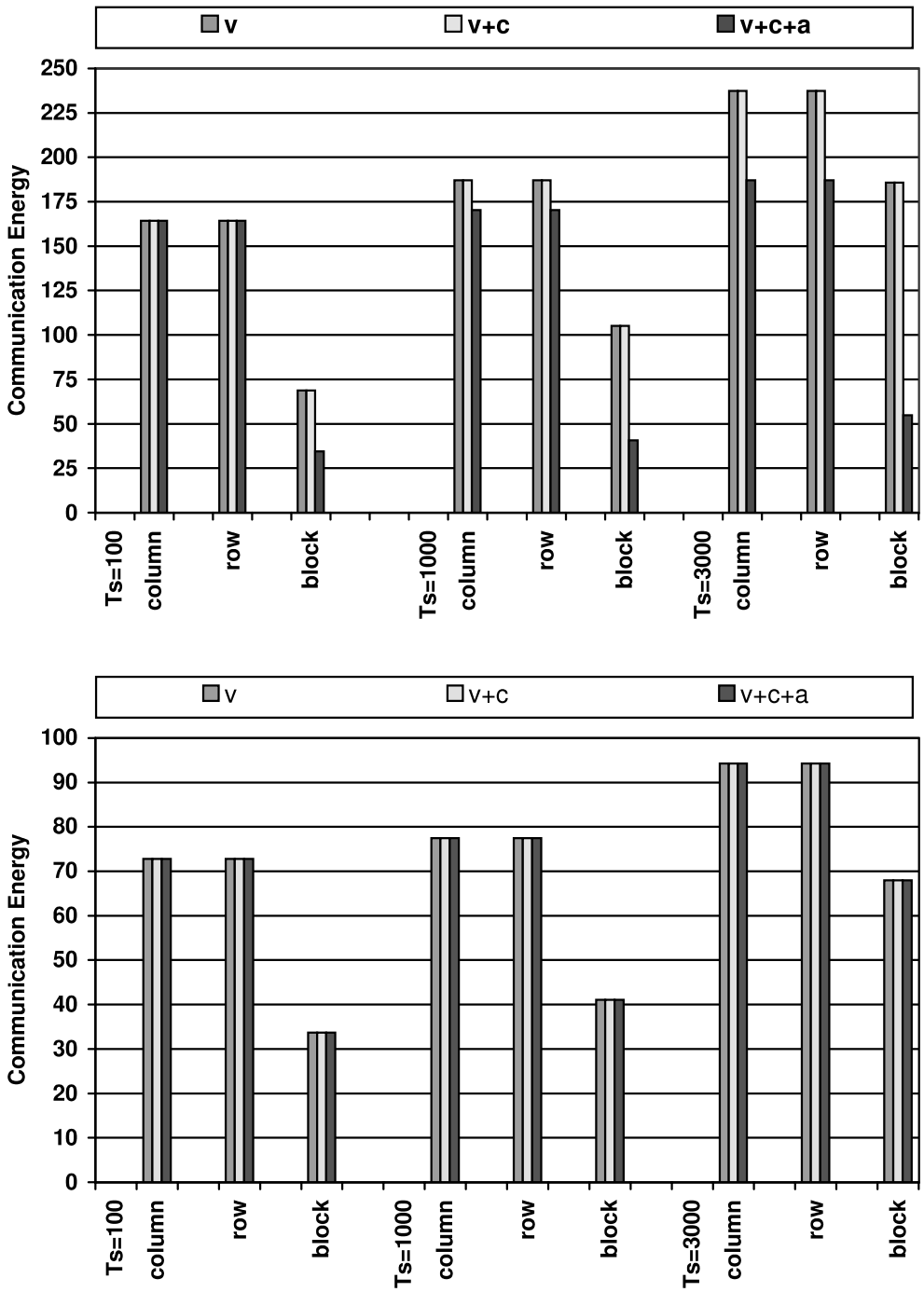


Figure 36.4. The effect of startup time on communication energy of `adi` (top) and `jacobi` (bottom).

rate (an extreme case), the communication energy is almost the same for column, row, and block decompositions.

Recall that our base configuration consists of 160 sensor nodes. To see the impact of changing the network size on energy consumption, we performed another set of experiments. In these experiments, we changed the network size from 40 nodes to 640 nodes at regular intervals. The first graph (top) in

© 2005 by Chapman & Hall/CRC

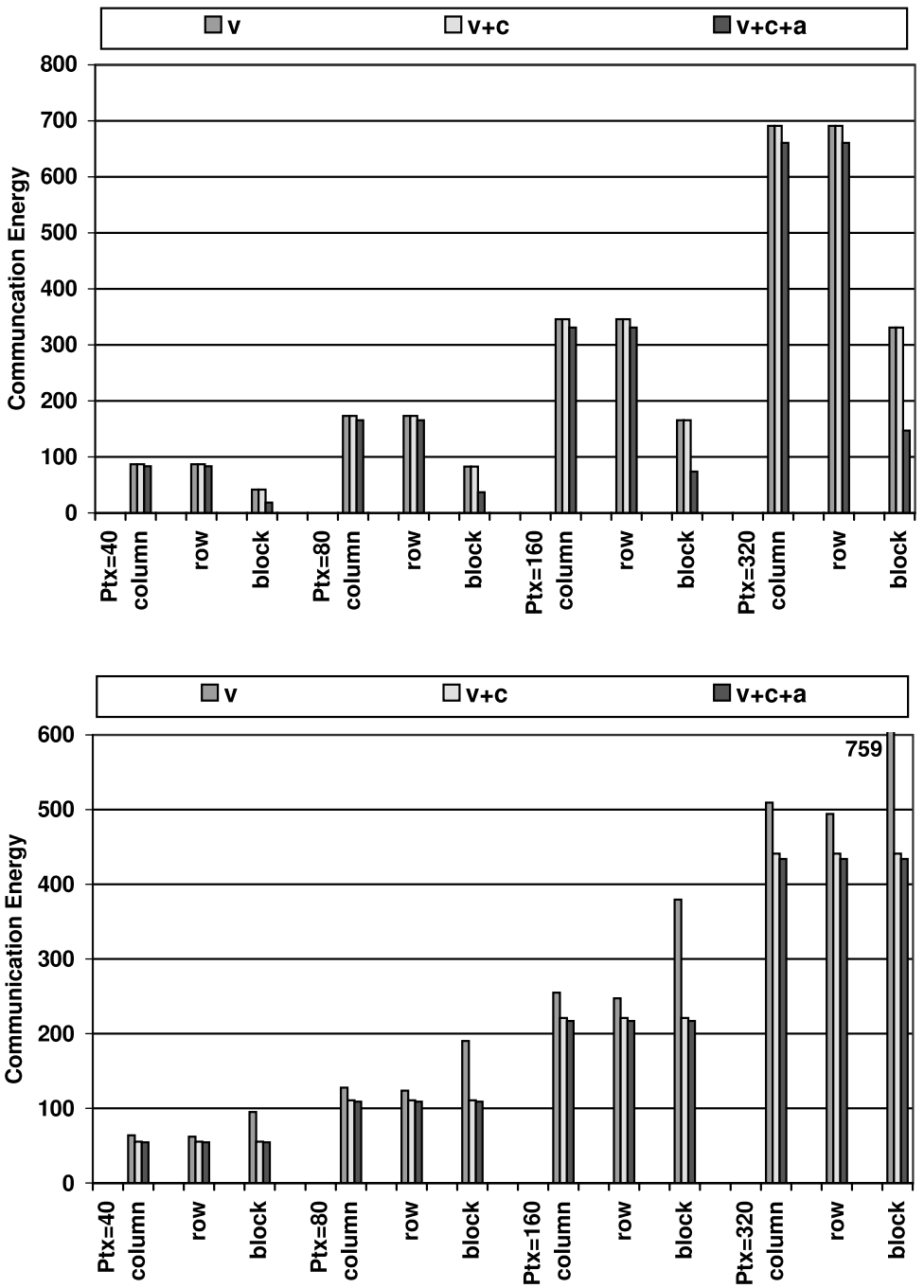


Figure 36.5. The effect of transceiver power on communication energy of `adi` (top) and `full-search` (bottom).

Figure 36.7 gives the communication energy consumption of `adi` when row (or column) decomposition is used. All other parameters are the same as in the base configuration (Table 36.2). We see that the communication energy increases as the network size is increased. This is due to the increase in the number of messages. However, if the data are decomposed in block manner, we obtain the energy behavior plotted in the bottom graph in Figure 36.7. We now observe that block decomposition

© 2005 by Chapman & Hall/CRC

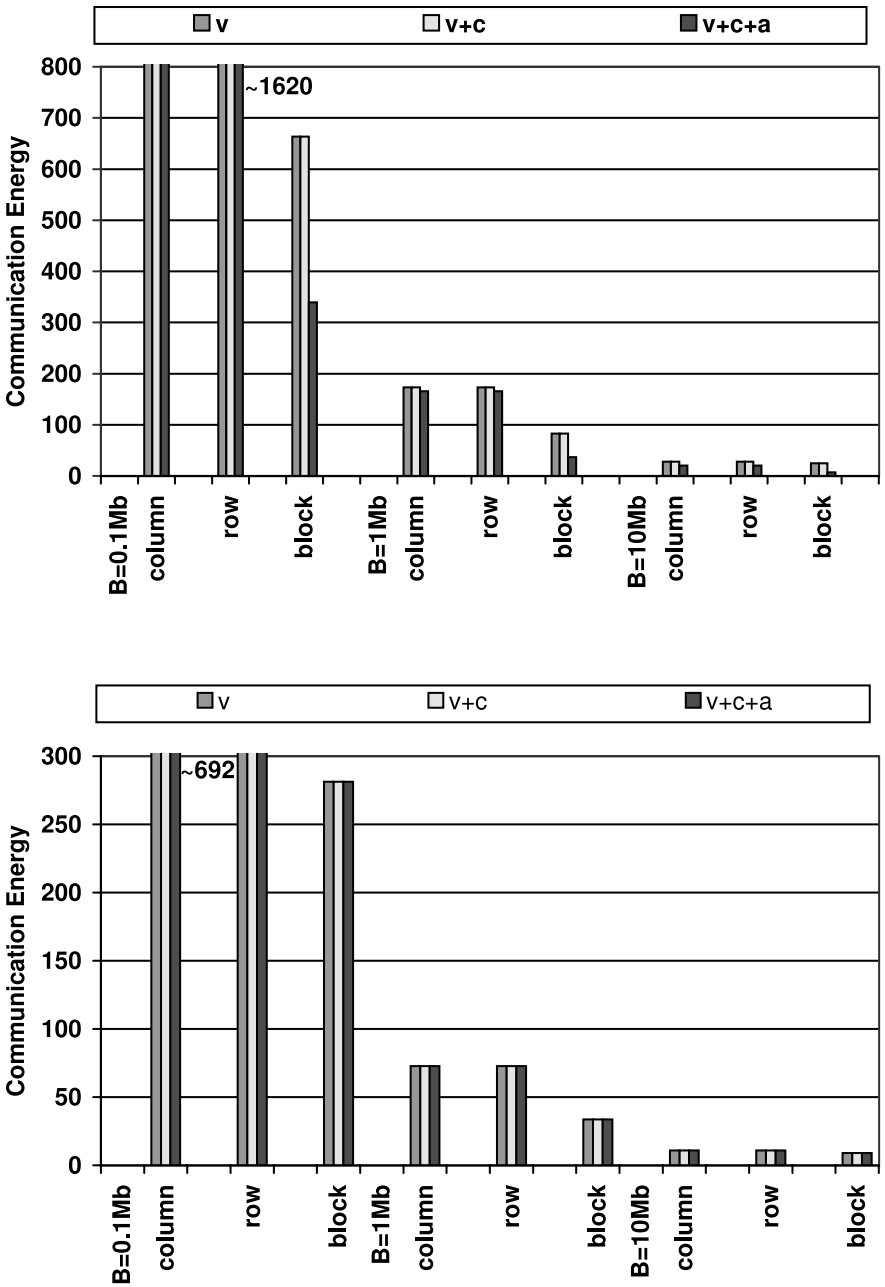


Figure 36.6. The effect of data transmit/receive rate on communication energy of `adi` (top) and `jacobi` (bottom).

in conjunction with message aggregation makes an important difference and limits the increase in communication energy. This is because, in this benchmark, the increase in number of messages is limited due to message aggregation. This example clearly demonstrates the importance of suitable combination of data decomposition and communication optimization. Therefore, we believe that future compilers/programming environments that target networks of microsensors should focus on reducing communication. Our experiments with other benchmark codes in our experimental suite also showed similar trends.

© 2005 by Chapman & Hall/CRC

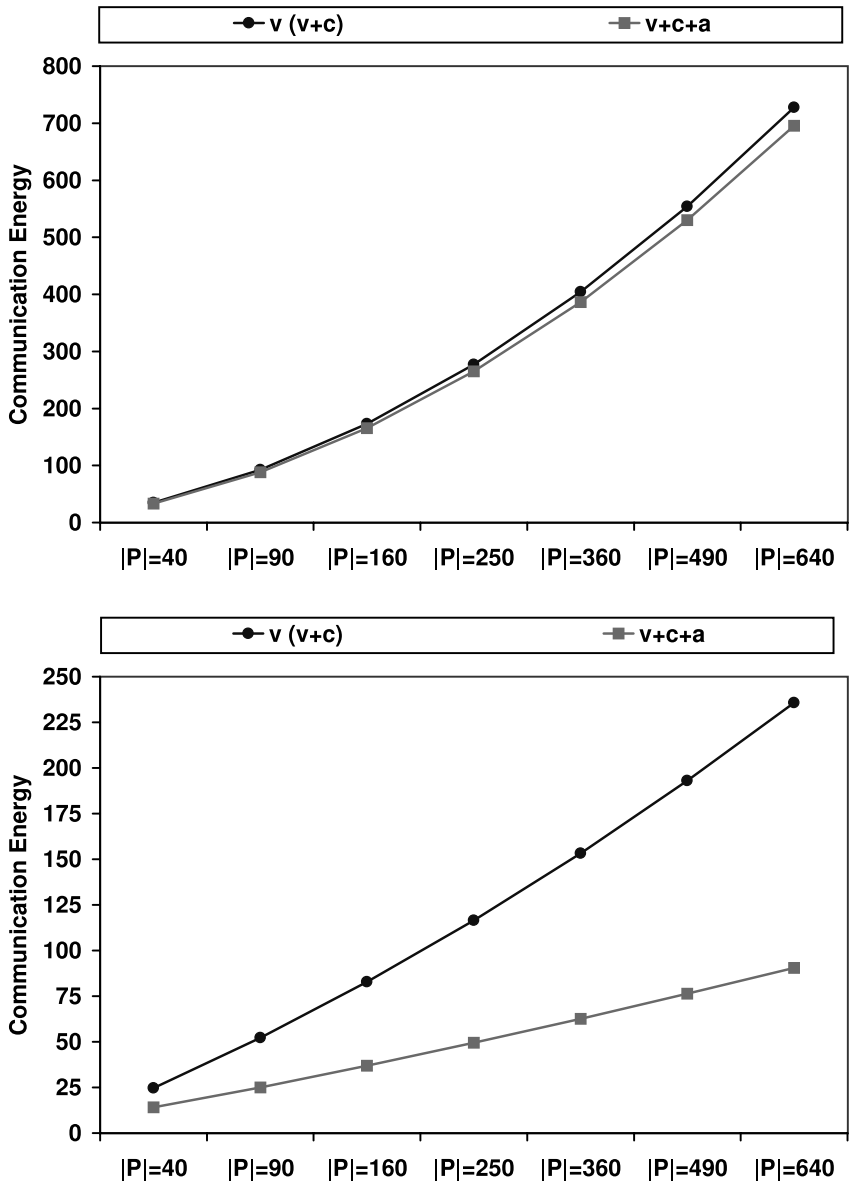


Figure 36.7. The effect of the number of sensor nodes on communication energy of `adi`. Top: row/column decomposition. Bottom: block decomposition.

36.5.3 Impact of Inter-Nest Message Optimization

As explained earlier, inter-nest optimizations try to eliminate the unnecessary communications by taking advantage of already communicated data. For example, if a set of array elements have already been communicated in a nest, they do not need to be communicated in the next nest unless they are modified between the two nests. The aggressiveness of such an optimization is measured in terms of the number of nests that can be considered at once. The least aggressive form considers only the neighboring nests, and does not try to take advantage redundant communication that happened, say, two nests earlier. Among our benchmark codes, `tomcatv` has an access pattern that can take advantage of this optimization. We experimented with five different versions of this benchmark. The first one (the aggressiveness level is 1) does not apply any inter-nest optimization, whereas the last one

© 2005 by Chapman & Hall/CRC

(the aggressiveness level is 5) can eliminate redundant data anywhere in the entire application code. As we move from level 1 to level 5, we use more and more inter-nest optimizations.

One important drawback of inter-nest optimizations is that the code after the transformations can become extremely complex. This is because applying inter-nest communication optimization requires hoisting a given communication in the code as much as possible so that it can be combined with communication coming from another nest (which will be executed earlier). As shown in previous research [25], this requires extensive inter-nest data dependence analysis (using a polyhedral tool such as Omega Library [26]) and code restructuring. This code restructuring (basically, it is a process of combining different array segments that will be communicated in a single message) can render the resulting (communication-optimized) code very complex, thereby increasing the energy spent during computation in datapath and memory. Therefore, when inter-nest optimization is employed, there is a trade-off between computation energy and communication energy.

To study this trade-off, in Figure 36.8 we give the computation and communication energies of `tomcatv` optimized using inter-nest optimizations with different levels of aggressiveness.

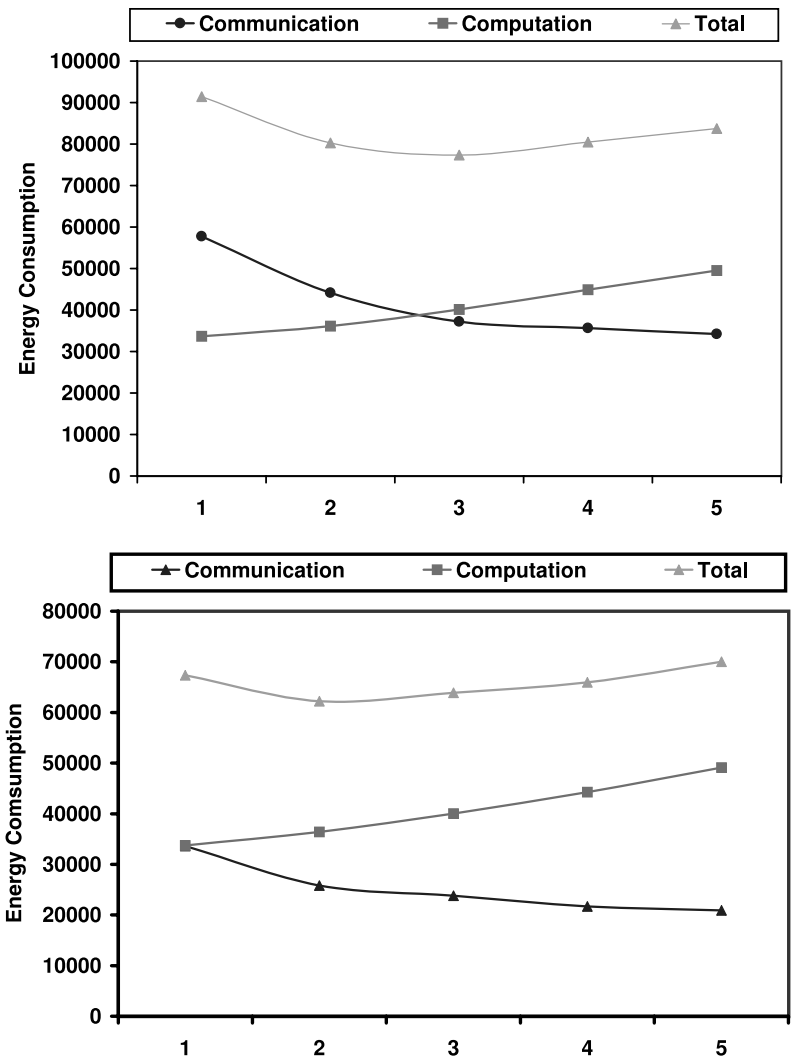


Figure 36.8. The effect of the inter-nest message optimizations on computation energy and communication energy of the `v+c+a` version of `tomcatv`. Top: row decomposition. Bottom: block decomposition.

© 2005 by Chapman & Hall/CRC

These graphs also show the overall (computation plus communication) energy (denoted Total). We see from these results that there is an operating point (in terms of aggressiveness) that gives the best results from the total energy viewpoint. In this example code, this optimum point is 3 when row decomposition is used, and 2 when block decomposition is used. Going beyond this point (i.e. trying to optimize communication energy even more aggressively) leads to so much increase in computation energy that it cannot be offset by the decrease in communication energy. Therefore, these results illustrate the trade-off between communication and computation energy. Based on these results, we can conclude that an optimizing compiler should try to determine this optimal operating (optimization) point to achieve the best results.

36.5.4 Impact of Overlapping Communication with Computation

The results presented so far were obtained under the assumption that a processor sits idle during communication. In fact, we assumed that the processor, when waiting for the data to be sent, places itself (or a tiny operating system does this for it) into a low-power mode where energy consumption is negligible. This is not very realistic, as processor and memories typically consume some amount of leakage during these waiting periods. Obviously, the longer the waiting period, the higher the leakage energy consumption. Consequently, we can reduce the leakage energy consumption by reducing the time that the processor waits idle for the communication to be completed. This can be achieved by overlapping the computation with communication. Consider the following code fragment:

```
for(i=2;i<=N-1;i++)
  for(j=2;j<=N-1;j++)
    u[i][j] = (v[i-1][j]+v[i+1][j]
              +v[i][j-1]+v[i][j+1])/4;
```

If we assume a `[*][block]` data decomposition for `u` and `v` across `M` sensor nodes, each node (except the first and the `Mth` one) needs one column of array `v` from its left neighbor and one column of array `v` from its right neighbor. Note that each processor owns an $N \times N/M$ portion of array `v` (and `u`). Since there are no data dependences in this code, the processor can execute its loop iterations in any order. Consequently, it can divide its loop iterations into three disjoint groups: (i) the iterations that are needed to compute the values of the elements to be sent; (ii) iterations that work only on local data (i.e. do not need any communication); (iii) the iterations that can be completed only after data have been obtained from its neighbors. Therefore, a given nest can be restructured as:

```
1. Execute iterations in (i)
2. Send data
3. Execute iterations in (ii)
4. Receive data
5. Execute iterations in (iii)
```

Note that this is very different from the straightforward execution, which can be summarized as:

```
1. Send data
2. Receive data
3. Execute iterations in (i+ii+iii) in their original order
```

Note also that in our current example above the first group (i) is an empty set. However, there will still be gains in using the restructuring above. As soon as a processor sends its data, instead of waiting for the data it needs (for executing iterations in (iii)), it continues with execution of iterations in (ii). Then, in step 4, it does not spend too much time. And finally, it executes the iterations in (iii). Note that, by separating the send and receive and doing some useful work between them, the message waiting time is effectively reduced. To evaluate the energy savings due to this optimization, we applied it to three

© 2005 by Chapman & Hall/CRC

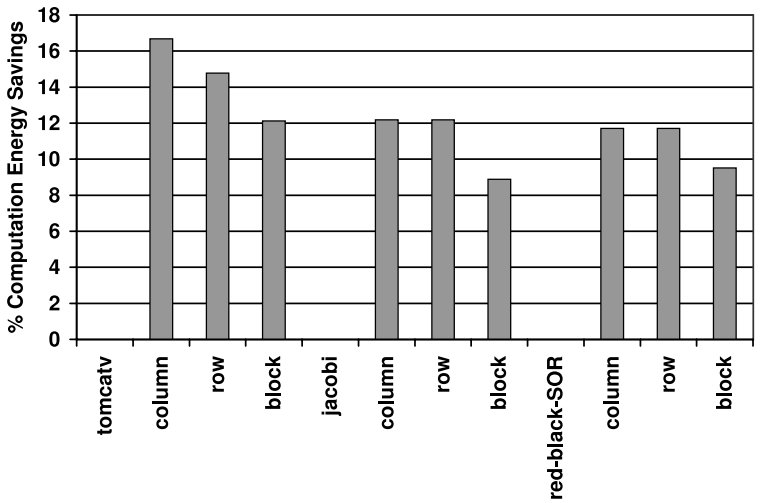


Figure 36.9. Percentage savings in computation energy due communication/computation overlapping.

benchmarks from our experimental suite, and measured the percentage computation energy savings. The results shown in Figure 36.9 indicate that our approach can save around 20% of the original computation energy of the `v+c+a` version. In these experiments, we have assumed that the leakage energy consumption per cycle of a component is 20% of the per access dynamic energy consumption of the same component. Since current trends indicate that leakage energy consumption will be more important in future, this optimization can be expected to be more useful with upcoming process technologies.

36.5.5 Communication Error

Communication errors in a sensor network may cause a significant energy waste due to retransmission. To see whether our optimizations better resist against this increase in energy consumption, we performed another set of experiments where we measured the energy consumption under a random error model. Since the results with most of the benchmarks are similar, here we focus only on `parallel-hier`. Figure 36.10 gives the communication energy consumption of the `v+c+a` version

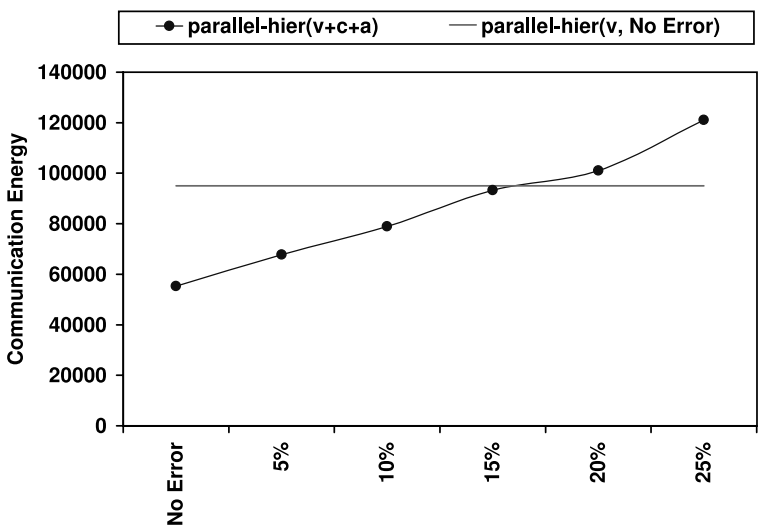


Figure 36.10. Impact of communication error.

© 2005 by Chapman & Hall/CRC

of this application assuming different error rates. For comparison purposes, we also give the communication energy consumption (in microjoules) of the `v` version without error. The results shown indicate that the `v+c+a` version starts to consume more energy than the error-free `v` version only beyond a 15% error rate. In other words, an application powered by our optimization can tolerate more errors (than the original application) under the same energy budget.

36.6 Our Compiler Algorithm

In this section we propose a compiler-based communication optimization algorithm for microsensor networks. Our algorithm combines the message vectorization, coalescing, aggregation, inter-nest optimization, and computation/communication overlapping in a unified framework. It takes as input a (sequential) C program annotated with data (array) decompositions. Its output is a parallelized node program that is to be executed in each sensor node. Each node program is parameterized using the `(a,b)` coordinates and contains optimized and automatically inserted communication calls (send/receive statements). Our algorithm is implemented using the SUIF compiler infrastructure [27] and performs the following steps:

- Using the owner-computes rule, each loop nest is parallelized. As explained earlier in the chapter, the owner-computes rule distributes loop iterations over sensor nodes in such a way that each sensor nodes executes iterations that assign values to the array elements it owns.
- Each loop nest is optimized using message vectorization, coalescing, and aggregation. Note that, since these transformations do not affect computation energy significantly, we can apply them as aggressively as possible. In the ideal case, the compiler tries to obtain the `v+c+a` version of each nest in the code.
- Communication energy is optimized using inter-nest optimization. However, since using this optimization aggressively can cause a large increase in computation energy, our compiler takes a different approach. It first uses the optimization by considering only the neighboring two nests. That is, it takes advantage of a previous communication if and only if the said communication occurs in the previous nest. The compiler then estimates the computation energy and communication energy. We estimate the computation energy using the publicly available energy-aware compilation framework presented by Kadayif *et al.* [28]. This compiler estimates the energy consumption at the source-level for our single-issue, five-stage pipelined processor. The energy consumed in a datapath is dependent on the number, types, and sequence of instructions executed. Consequently, using the approach of Wolf *et al.* [29], the compiler estimates this information and calculates the datapath energy consumption. The energy consumed in SRAM memories is dependent on the number of data and instruction accesses and the memory configuration (i.e. capacity, number of ports, etc.). The compiler estimates the number of instruction and data accesses (for our simple architecture) and then, using the configuration parameters, computes the energy that will be expended in memory. The energy consumed in a single cycle due to the clock network depends on the parts of the clock network that are active. The PLL and the main clock distribution circuitry are normally active every clock cycle during execution. Therefore, the compiler captures the energy consumption due to those two components by estimating the number of cycles that the code would take. However, the participation of the clock-load varies based on the active components of the circuit, as determined by the software executing on the system. For example, the clock to the SRAMs is gated (disabled) when they are not used. The compiler exploits the estimation techniques for the datapath and memories explained above to account effectively for this varying clock-load in a given cycle. This compiler-based energy estimation framework has been validated using the cycle-accurate architectural-level energy simulator (SimplePower), and found to be within 6% error margin while providing significant estimation speedup. Note that, since our processor is very simple (which is suitable to be used in a sensor node), such a compiler-based, high-level energy estimation is possible.

© 2005 by Chapman & Hall/CRC

Estimating communication energy within the compiler, on the other hand, involves determining the number of messages and the number of elements to be sent and received. The compiler extracts these data from the code (considering the optimization applied) and then using the analytical formulations from Shih *et al.* [7], along with energy parameters (as those in our base configuration), estimates the communication energy. The overall energy estimation is, therefore, the sum of the computation energy and communication energy estimations. The compiler then compares this estimate with the estimate obtained from the `v+c+a` versions of nests (i.e. without any inter-nest optimization), and checks whether applying inter-nest optimization reduces the *overall* (program-wide) energy consumption. If it does not, then this step in the algorithm stops, and the compiler proceeds with the next step. On the other hand, if it does, then the compiler increases the aggressiveness of inter-nest optimization (i.e. it considers three neighboring nests at a time) and repeats the energy estimation. If this last energy estimation is smaller than the previous one (i.e. the one obtained considering only two nests at a time), then the compiler continues to increase the level of aggressiveness, and so on; otherwise it terminates this step. In this way, the highest aggressiveness level with the minimum overall energy estimation is found. For example, this strategy successfully determines the optimum aggressiveness levels shown in Figure 36.8.

- Each loop nest is checked to see whether it can benefit from overlapping communication with computation. To do this, the compiler generates the sets (i), (ii), and (iii) as discussed in Section 36.5.4. If the set (ii) is not empty, then this means that the nest in question can benefit from overlapping. If this is the case, then the compiler restructures the nest accordingly.

Now, we would like to discuss three important points briefly. First, the algorithm discussed above works with a user-specified data decomposition. However, it is possible to add an outermost loop to this algorithm to enumerate some subset of possible data decompositions, and estimate energy consumption of optimizations under different decompositions. In this way, the compiler can determine the most suitable data decompositions from the energy perspective. Second, when we used the algorithm described above for optimizing communication energy of our benchmark codes, we found that, for a given (user-specified) data decomposition, this algorithm determines the optimized code with minimum overall energy consumption. Third, as mentioned earlier, to implement inter-nest optimization, our approach uses a polyhedral tool [26], which might take a significant amount of compilation time. To check this, we measured the time spent in compilation of each benchmark. We found that, when the optimization algorithm described above is used, the time spent in compilation varies between 3.4 and 5.1 s, averaging (across all benchmarks) in 4.4 s. Considering the large energy benefits at run time, we believe that these compilation times are tolerable. Based on this discussion, we believe that this compiler algorithm can be part of a software framework for optimizing applications in sensor networks.

36.7 Conclusions and Future Work

Advances in CMOS technology and microsensors enabled construction of large, power-efficient networked sensors. The energy behavior of applications running on these networks is largely dictated by the software support employed, such as operating systems and compilers. Our results presented in this chapter indicate that high-level communication optimizations can be vital if one wants to keep the communication energy consumption of a sensor network under control. Our results also show that it is possible to adopt a unified compiler algorithm that combines message vectorization, coalescing, aggregation, inter-nest optimizations, and computation/communication overlapping.

While these optimizations are vital for reducing communication energy, there are also optimizations [2,10,20,30,31] that can target computation energy expended in sensor nodes. Our future work will study the interaction between communication and computation optimizations. Another promising research direction is implementation of efficient software-level communication primitives for sensor networks. Such primitives can improve the effectiveness of the compiler and, at the same time, can lead to a better optimized code.

© 2005 by Chapman & Hall/CRC

References

[1] Hill, J. *et al.*, System architecture directions for network sensors, in *Proceedings of ASPLOS*, 2000.

[2] Min, R. *et al.*, Low-power wireless sensor networks, in *Proceedings of VLSI Design'2001*, January 2001.

[3] Chen, B. *et al.*, Span: an energy-efficient coordination algorithm for topology maintenance in *ad hoc* wireless networks, in *Proceedings of the ACM MOBICOM Conference*, Rome, Italy, July 2001.

[4] Asada, G. *et al.*, Wireless integrated network sensors: low power systems on a chip, in *Proceedings of ESSCIRC'98*, Hague, Netherlands, September 22–24, 1998.

[5] Rabaey, J. *et al.*, PicoRadio supports *ad-hoc* ultra-low power wireless networking, *IEEE Computer Magazine*, July, 42, 2000.

[6] Lim, A., Distributed services for information dissemination in self-organizing sensor networks. Special Issue on Distributed Sensor Networks for Real-Time Systems with Adaptive Reconfiguration, *Journal of Franklin Institute*, 338, 707, 2001.

[7] Shih, E. *et al.*, Physical layer driven protocol and algorithm design for energy-efficient wireless sensor network, in *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, Rome, Italy, July 16–22, 2001.

[8] Cho, S.-H. and Chandrakasan, A., Energy-efficient protocols for low duty cycle wireless microsensor networks, in *Proceedings of ICASSP'2001*, May 2001.

[9] Ye, W. *et al.*, An energy-efficient MAC protocol for wireless sensor networks, in *Proceedings of the 21st International Annual Joint Conference of the IEEE Computer and Communications Societies*, New York, NY, USA, June 2002.

[10] Catthoor, F. *et al.*, *Custom Memory Management Methodology — Exploration of Memory Organization for Embedded Multimedia System Design*, Kluwer Academic Publishers, 1998.

[11] Chandrakasan, A. *et al.*, *Design of High-Performance Microprocessor Circuits*, IEEE Press, 2001.

[12] Wang, Y. *et al.*, *Multimedia Signal Processing*, IEEE Press, 1997.

[13] Wang, D.J. and Hu, Y.H., Multiprocessor implementation of real time DSP algorithms, *IEEE Transasctions on VLSI Systems*, 3(3), 393, 1995.

[14] Sinha, A. and Chandrakasan, A., Operating system and algorithmic techniques for energy scalable wireless sensor networks, in *Proceedings of the 2nd International Conference on Mobile Data Management*, January 2001.

[15] Brooks, R.R. and Iyengar, S.S., *Multi-Sensor Fusion: Fundamentals and Applications with Software*, Prentice Hall, 1998.

[16] Wolfe, M., *High Performance Compilers for Parallel Computing*, Addison-Wesley Publishing Company, CA, 1996.

[17] Kadayif, I. *et al.*, An energy-oriented evaluation of communication optimizations for microsensor networks, in *Proceedings of International Conference on Parallel and Distributed Computing*, Klagenfurt, Austria, August 2003.

[18] Bozkus, Z. *et al.*, Compiling HPF for distributed memory MIMD computers, in *The Interaction of Compilation Technology and Computer Architecture*, Lilja, D. and Bird, P. (eds), Kluwer Academic Publishers, 1996.

[19] Bozkus, Z. *et al.*, Compiling distribution directives in a Fortran 90D compiler, in *Proceedings of the 5th IEEE Symposium on Parallel and Distributed Processing*, December 1993.

[20] Benini, L. and De Micheli, G., System-level power optimization: techniques and tools, *ACM TODAES*, 5(2), 115, 2000.

[21] Vijaykrishnan, N. *et al.*, Energy-driven integrated hardware-software optimizations using Simple Power, in *Proceedings of the International Symposium on Computer Architecture*, June 2000.

[22] Chen, R.Y. *et al.*, Validation of an architectural level power analysis technique, in *Proceedings of the 35th Design Automation Conference*, June 1998.

© 2005 by Chapman & Hall/CRC

[23] Cmelik, B. and Keppel, D., Shade: a fast instruction-set simulator for execution profiling, in *Proceedings of the 1994 ACM SIGMETRICS Conference on the Measurement and Modeling of Computer Systems*, May 1994, 128.
[24] Wilton, S. and Jouppi, N.P., CACTI: an enhanced cycle access and cycle time model, *IEEE Journal of Solid-State Circuits*, 31(5), 677, 1996.
[25] Kandemir, M. *et al.*, A global communication optimization technique based on data-flow analysis and linear algebra, *ACM Transactions on Programming Languages and Systems*, 21(6), 1251, 1999.
[26] Kelly, W. *et al.*, The Omega library interface guide, Technical Report CS-TR-3445, CS Dept., University of Maryland, College Park, MD, March 1995.
[27] Amarasinghe, S.P. *et al.*, The SUIF compiler for scalable parallel machines, in *Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing*, February 1995.
[28] Kadayif, I., *et al.*, EAC: a compiler framework for high-level energy estimation and optimization, in *Proceedings of the 5th Design Automation and Test in Europe Conference*, Paris, France, March 2002.
[29] Wolf, M. *et al.*, Combining loop transformations considering caches and scheduling, in *Proceedings of the International Symposium on Microarchitecture*, Paris, France, December 1996, 274.
[30] Benini, L. *et al.*, Synthesis of application-specific memories for power optimization in embedded systems, in *Proceedings of DAC'00*, 2000, 300.
[31] Simunic, T., Dynamic voltage scaling and power management for portable systems, in *Proceedings of DAC'01*, 2001, 524.

© 2005 by Chapman & Hall/CRC

37

Sensor-Centric Routing in Wireless Sensor Networks*

Rajgopal Kannan and S.S. Iyengar

37.1 Introduction

Sensor networks are an emerging and critically important technology area for the acquisition of spatio-temporally dense data in hazardous and unstructured environments [1]. With applications such as environmental and seismic monitoring, these networks represent a mechanism by which information technology can be used to mitigate the vulnerabilities of society with regard to natural and man-made catastrophic events. However, there are several obstacles to be overcome in order to effectively manage and utilize the large amounts of collected sensor data in these networks.

1. Sensors have limited and unreplenishable power resources, making energy management a critical issue. In particular, routing protocols must be energy-aware and designed to prolong the lifetime of individual sensors (and indirectly network lifetime in terms of network connectivity/information utility). When a sensor receives a packet to be forwarded, the selection of the next-hop node must be based in part on the communication energy costs. For example, routing on the basis of minimizing aggregate energy costs on the path is one possible metric [2–4].
2. Sensors are unattended. Nodes must make decisions independently, without recourse to a central authority, because of the energy needed for global communication and latency of centralized processing. In particular, sensors must have the capacity to decide independently whether to participate in a routing path, and if so to select the next-hop destination based on some (local) energy considerations.

Clearly, the untethered and unattended nature of sensors constrain their actions as individual devices, since they must independently and efficiently utilize their limited energy resources. However, designing sensor network solutions that only optimize energy consumption will not always lead to efficient architectures, since the above constraints do not account for collaborative trade-offs between groups of sensors. Note that collaborative interaction among sensors provides some network-wide benefits (as opposed to "energy" benefits to individual sensors), where network-wide is a semantic term

*Some of the material in this chapter is excerpted from [15] and reprinted with the permission of Elsevier.

© 2005 by Chapman & Hall/CRC

referring to overall goals of the entire network or to a sufficiently large group of sensors. Consider, for example, collaborative data mining/information fusion among sensors to respond meaningfully to queries [5,6]. Too many sensors simultaneously participating in the collaborative decision making required for aggregation of mined data will lead to excessive routing paths in the network, thereby increasing energy consumption and competition for communication resources. On the other hand, too little collaborative data aggregation will make distributed mining inaccurate and ineffective. Thus, sensors are implicitly constrained by a third factor: to increase information utilization of the network, sensors must cooperate to maximize network-wide objectives while maximizing their individual lifetimes. We label this paradigm for broad sensor network operation as *sensor-centric*.

The choices for untethered and unattended sensors under this paradigm are a natural fit for a game-theoretic framework in which sensors are modeled as rational/intelligent agents cooperating to find optimal network architectures that maximize their payoffs in a network game, where sensor payoffs are defined as benefits to the network of this sensor's action minus individual costs (as opposed to aggregate path costs). This sensor-centric paradigm is general enough to model sensor payoffs under a variety of network game scenarios, such as security, reliability, or delay. For example, Kannan and Iyengar [7] show how a network game with aggregate route measures such as path-length/delay can be used to derive optimal routes in polynomial time for certain classes of sensor networks. This chapter provides a sensor-centric model of reliable routing interactions between sensors.

37.2 Sensor-Centric Reliable Routing

There are many popular datacentric [8] routing algorithms for minimizing energy consumption, such as MECN [9] and diffusion routing [10], which use local gradients to identify paths for sending information. GEAR [4] uses an energy-aware metric and also the geographical position of each node to determine a route. Sohrabi *et al.* [11] and Shah and Rabaey [12] describe routing algorithms for sensor networks that take energy constraints and quality-of-service considerations into account. Shah and Rabaey [12] show that the lowest energy path may not always be optimal for long-term network connectivity. In general, routing algorithms that attempt to minimize overall energy consumption costs may result in uneven energy consumption patterns across sensor nodes. Consequently, some nodes could deplete their energy resources sooner than necessary, thereby reducing the information utility of the sensor network.

While the energy-efficiency of routes is an important parameter, maximizing information utility and network lifetime implies that the *reliability* of a data transfer path from reporting to querying sensor is also a critical metric. This is especially true given the possibility of sensor failure in hazardous deployments and the susceptibility of sensor nodes to denial-of-service (DoS) attacks and intrusion [13,14]. We therefore require a sensor-centric model for reliable energy-constrained routing in which sensors route over the most reliable paths while minimizing their own power/energy consumption, rather than some aggregate path energy criterion. In effect, each sensor independently assumes itself as critical to the network's survival and, therefore, attempts to reduce its energy costs, while still satsifying network-wide objectives.

The sensor-centric paradigm of reliable energy-constrained routing has two intuitive benefits. First, it is in the interests of long-term network operability that nodes survive even at the expense of somewhat longer (but not excessively so!) paths. The network will be better served when a critical sensor can survive longer by transmitting via a cheaper link rather than a much costlier one for a small gain in reliability or delay. Second, it takes the cost distributions of individual sensors into account while choosing good paths. The advantages of modeling rational, self-interested sensors can be seen easily from the following example. Given a path involving three sensors with absolute communication costs in the low, medium and high ranges respectively, choosing a reliable path subject to minimzing overall costs might lead to the first two nodes having to select their highest cost links, as the third node is dominant in the overall cost. This would run counter to the long-term operability goal of the network.

© 2005 by Chapman & Hall/CRC

37.3 Reliable Routing Model

We now describe a model for reliable data-centric routing with data-aggregation in sensor networks taken from Kannan *et al.* [15]. In data-centric routing, interest queries are disseminated through the network to assign sensing tasks to sensor nodes. Responses from sensors are aggregated at intersecting nodes to reduce data implosion and overlap. With data aggregation, the sensor network can be perceived as a reverse multicast tree with information fused at intersecting nodes and routed to the sink node at the root.

The problem of reliable query routing (RQR) in a sensor network can be defined as follows. Given that data transmission in the network is costly and nodes are not completely reliable, how can we induce the formation of a maximally reliable data aggregation tree from reporting sensors (sources) to the query originating node (sink), where every sensor is 'smart', i.e. it can trade off individual costs with network-wide benefits? This optimally reliable data aggregation tree (henceforth the optimal RQR tree) will naturally be distinct from standard multicast trees, such as the Steiner tree [16] or shortest path trees, which minimize overall network costs, and therefore cannot represent the outcome of self-interested sensors. The solution to this problem lies in designing a routing game with payoff functions, such that its Nash equilibrium [17] corresponds to the optimal RQR tree.

Let $S = \{s_1, \ldots, s_n\}$ denote the set of sensors, modeled as players in the routing game. Assume that a query has been sent from the sink node $s_q = s_n$ to the nodes in S. The query may match the attributes of data stored at each s_i to varying degrees. We abstract this idea of information retrieval by attaching a value $v_i \in \Re$ to the data retrieved from each sensor s_i, $1 \leq i < n$ ($v_i = 0$ for nodes whose sensor data does not satisfy the specified attributes of the query). This will allow popular (high value) data items to be routed over more reliable paths, even at higher costs. Information is routed to s_q through an optimally chosen set $S' \subseteq S$ of sensors. Communication between neighboring sensors i and j is implemented via an underlying medium access control (MAC) protocol with associated distance-based transmission energy cost $c_{ij} > 0$ (packet reception is assumed costless here for simplicity). Note that alternate link cost metrics such as delay at the next node or link cost inversely proportional to remaining battery life can also be used. The model also requires the values of path reliability which can be measurement based, using periodic observation of DoS patterns with statistical inference tools. For simplicity, we assume that node s_i can independently fail with a probability $(1 - p_i) \in [0, 1)$ ($p_q = 1$). These features of the model allow sensors to decide rationally (by computing individual payoffs) whether or not to participate in routing data of a given significance. Link formation in the network occurs by a process of simultaneous reasoning at each node, leading to a path from each s_i with nonzero value v_i to s_q. It can be shown for this particular game that sequential reasoning by nodes in order of selection will also produce exactly the same equilibrium paths.

Thus the graph $G = (S, E, P, C)$ represents an instance of a data-centric sensor network in which data of value v_i are to be optimally routed from node s_i to node s_q, with S the set of sensors interconnected by edge set E, $P(s_i) = p_i$ the node success probabilities, and $C(s_i, s_j) = c_{ij}$ the cost of links in E.

There are several possible ways to model payoffs to sensor nodes, resulting in different query reporting architectures, as shown by Kannan and Iyengar [7]. Here, we describe the different components of a strategic RQR game with a simple reliability payoff model.

Strategies. Each node's strategy is a binary vector $l_i = (l_{i1}, l_{i2}, \ldots, l_{ii-1}, l_{ii+1}, \ldots, l_{in})$, where $l_{ij} = 1$ ($l_{ij} = 0$) represents sensor s_i's choice of sending/not sending a data packet to sensor s_j. Since a sensor typically relays a received data packet to only one neighbor, we assume that a node forms only one link for a given source and destination pair of leader nodes. In general, a sensor node can be modeled as having a mixed strategy [17], i.e. the l_{ij} are chosen from some probability distribution. However, in this paper we restrict the strategy space of sensors to only pure strategies. Furthermore, in order to eliminate some trivial equilibria (such as all paths with no short-circuits, the empty network, etc.), each sensor's strategy is constrained to be nonempty and strategies resulting in a node linking to its ancestors (i.e. routing loops) are disallowed. Consequently, the strategy space of each sensor s_i is such that $\text{Prob}[l_{ij} = 1] = 1$ for exactly one sensor s_j and $\text{Prob}[l_{ij} = 1] = 0$ for all other sensors, such that no

© 2005 by Chapman & Hall/CRC

routing loops are formed. Under these assumptions, each meaningful strategy profile $l = (l_1, \ldots, l_n)$ becomes a reverse tree $\mathcal{T}$, rooted at the sink s_q. We now proceed to model the payoffs in this game.

Payoffs. Consider a strategy profile $l = (l_i, l_{-i})$ resulting in a tree $\mathcal{T}$ rooted at s_q, where l_{-i} denotes the strategy chosen by all the other players except player i. Since every sensor that receives data has an incentive in its reaching s_q, the benefit to any sensor s_i on $\mathcal{T}$ must be a function of the path reliability from s_i onwards. Since the network is unreliable, the benefit to player s_i should also be a function of the expected value of information at s_i. Hence, we can write the payoff at s_i as

$$\Pi_i(l) = \begin{cases} g_i(v_1, \ldots, v_{n-1})R_i - c_{ij} & \text{if } s_i \in \mathcal{T} \\ 0 & \text{otherwise} \end{cases}$$

where R_i denotes the path reliability from s_i onwards to s_q, and g_i is the expectation function, which is explained below.

Let $\mathcal{V}_i = g_i(v_1, \ldots, v_{n-1})$ denote the expected value of the data at a node i and $F(i)$ the set of its parents. Then $\mathcal{V}_i = v_i + \sum_{j \in F(i)} p_j \mathcal{V}_j$, i.e. s_i gets information from its parents only if they survive with the given probabilities. The expected benefit to sensor s_i is given by $\mathcal{V}_i R_i$, i.e. i's benefits depend on the survival probability of players from i onwards. Hence, the payoff to s_i is $\Pi_i = R_i \mathcal{V}_i - c_{ij}$.

Definition 37.1. *A strategy l_i is said to be a **best response** of player i to l_{-i} if*

$$0 \leq \Pi_i(l_i, l_{-i}) \geq \Pi_i(l_i', l_{-i}) \quad \textit{for all} \quad l_i' \in \mathcal{L}_i$$

Let $BR_i(l_{-i})$ denote the set of player i's best response to l_{-i}. A strategy profile $l = (l_1, \ldots, l_n)$ is said to be an *optimal RQR tree* $\mathcal{T}$ if $l_i \in BR_i(l_{-i})$ for each i, i.e. sensors are playing a Nash equilibrium. In other words, the payoff to a node on the optimal tree is the highest possible, given optimal behavior by all other nodes. A node may get higher payoffs by selecting a different neighbor on another tree; however, it can only do so at the cost of suboptimal behavior (i.e. reduced payoffs to) by some other node(s).

Note that under the definitions above, although each sensor can form only one link, multiple equilibrium trees can exist.[1] Thus, the optimal strategy requires each node to select that node as next neighbor, the optimal tree through which it gets the highest payoff.

37.4 Results

This section contains results on two aspects of the RQR problem. We first analyze the complexity of computing the optimally reliable (or equilibrium) data aggregation tree in a given sensor network. This is followed by some analytical results that establish congruence between the optimal RQR path and other well-known path metrics, such as the most reliable path and other energy-conserving paths.

37.4.1 Complexity Results

We begin with the following general result.

Theorem 37.1. *Given an arbitrary sensor network G with sensor success probabilities P, communication costs C, and data of value $v_i \geq 0$ to be routed from each sensor s_i to the sink s_q, computing the optimaly reliable data aggregation tree $\mathcal{T}$ (the RQR tree) is NP-hard.*

Proof. Given any solution $\mathcal{T}'$ to the RQR problem, verifying the optimality of the successor for each node in $\mathcal{T}'$ requires exhaustively checking payoffs via all possible trees to s_q. Thus, RQR does not belong

[1] In the case of routing paths, payoff ties at a node can be broken by selecting the edge that leads to higher reliability. However, this is not always possible in the case of trees.

© 2005 by Chapman & Hall/CRC

to *NP*. That the RQR problem is *NP*-hard follows by reduction, using the following lemma which considers the special case of finding an optimal path, given a single source. (Note that this is equivalent to finding routing trees without data aggregation.) ■

Lemma 37.1. *Let $\mathcal{P}$ be the optimal RQR path for routing data of value v_r from a single reporting sensor s_r to the sink node s_q in a sensor network G where $v_i = 0 \quad \forall i \neq r$. Computing $\mathcal{P}$ is NP-hard.*

The proof is obtained by reduction from Hamiltonian path as shown by Kannan *et al.* [15]. The RQR path and tree problems remain *NP*-hard for the special case when nodes have equal success probabilities. However, the case when all edges have the same cost is much simpler, as shown below.

37.4.2 Analytical Results

Given the complexity of computing the optimal RQR tree, we try to analytically derive conditions that establish congruence between the optimal and other well-known, easily computable trees, such as the most reliable tree (i.e. the union of the most reliable paths) from sources to the sink and energy-conserving trees. Identifying these conditions on network parameters will save the overhead of computing optimal (or approximately optimal) RQR trees in these cases. For simplicity, we present these congruence results in terms of paths from a single source to the sink; the results can be easily extended to trees.

Let G be an arbitrary sensor network with a single-source node having data of value v_r ($v_i = 0$ for all other nodes). Then the following results hold. Note that the results describe only sufficient conditions for congruence with the optimal path.

Observation 37.1. *Given $p_i \in (0, 1]$ and $c_{ij} = c$ for all ij, then the most reliable path (tree) always coincides with the optimal RQR path (tree). For uniform p_i, the equilibrium RQR path is also the path with least overall cost.*

Before proceeding further, we now introduce some notation. For any node s_i, let $c_i = \{c_{ij}\}$, $c_i^{\max} = \max\{c_{ij}\}$ and $c_i^{\min} = \min\{c_{ij}\}$. Also, $c^{\max} = \max_i\{c_{\max}^i\}$ and $c^{\min} = \min_i\{c_{\min}^i\}$. We use $\mathcal{P}_i^l$ to denote a path of length l from s_i to s_q and benefits along this path by $\mathcal{P}_i^l$.

Proposition 37.1. *Given G and $P(s_i) = p \in (0, 1]$, for all i, the most reliable path from s_r to s_q will also be the optimal path if*

$$c_i^{\max} - c_i^{\min} < v_r p^m (1 - p)$$

for all s_i on the most reliable path $\mathcal{P}_r^m$.

Proof. Consider an arbitrary node s_i at a distance i from s_r. Since we have uniform p, reliability is now inversely proportional to path length. Let l be the length of the shortest path from s_i to s_q, on which s_{i+1} is the next neighbor of s_i. For s_i, $\mathcal{P}_i^l$ is optimal if

$$v_r p^{i+l} - c_{ii+1} > v_r p^{i+l+\lambda} - c_{ij} \quad \lambda = 1, 2, \ldots$$

$$\Rightarrow \frac{c_{ij} - c_{ii+1}}{v_r} < p^{i+l}(1 - p^{\lambda})$$

where s_j is a neighbor of s_i through which there is a simple path of length $l + \lambda$. Since $m = i + l$ on $\mathcal{P}_r^m$, the reliability term above is minimized for $\lambda = 1$, whereas the cost term is maximized at $c_{\max}^i - c_{\min}^i$. ■

© 2005 by Chapman & Hall/CRC

Note that the above result identifies sufficient constraints on costs for the most reliable path to also be optimal. The result shows that while the most reliable path can be costlier than other paths, to be optimal it cannot be "too" much more expensive. From the above result, it also follows that when $c^{\max} - c^{\min} < \quad p^m(1-p)$ this path coincides with the optimal, thereby providing a global bound on costs for congruence. The equivalent result for the most reliable tree can be obtained by substituting $\mathcal{V}_i$, the expected aggregated data value at s_i, for v_r in the above proposition.

We now look at the situation when the probabilities of node survival are nonuniform. Let s_i and s_{i+1} be subsequent nodes on the most reliable path. Denote by R_i, the reliability of the most reliable path from s_i to s_q, with R_i' being the reliability along any alternative path from s_i. Let $\Delta c_i = c_{ii+1} - c_{ij}$ where s_j is any neighbor not on the optimal path and ΔR_i is defined similarly.

Proposition 37.2. *Given G and $P(s_i) = p_i \in (0, 1]$, the most reliable path from s_r to s_q will be optimal if*

$$\frac{\Delta c_{i+1}}{\Delta c_i} < \frac{\Delta R_{i+1}}{\Delta R_i}$$

for all s_i and s_{i+1} on the most reliable path.

Proof. Let $\overline{R}_i$ represent the reliability on the portion of the most reliable path $\overline{\mathcal{P}}$ from s_r to s_i. Since $\overline{\mathcal{P}}$ is optimal, s_i cannot benefit by deviating if

$$\begin{aligned} v_r\overline{R}_i R_i - c_{ii+1} &> v_r\overline{R}_i R_i' - c_{ij} \\ \Rightarrow v_r\overline{R}_i &> \frac{\Delta c_i}{\Delta R_i} \end{aligned}$$

It follows that $v_r\overline{R}_{i+1} > \Delta c_{i+1}/\Delta R_{i+1}$. Since $\overline{R}_{i+1} = p_{i+1}\overline{R}_i$, we have $v_r p_{i+1}\overline{R}_i > \Delta c_{i+1}/\Delta R_{i+1}$. This can be rewritten as $1 \geq p_{i+1} > (\Delta c_{i+1}/\Delta c_i)(\Delta R_i/\Delta R_{i+1})$, which gives us $\Delta c_{i+1}/\Delta c_i < \Delta R_{i+1}/\Delta R_i$ as desired. ■

The easiest way to interpret this result is by rearranging the terms so that we can write it as $\Delta c_{i+1}/\Delta R_{i+1} < \Delta c_i/\Delta R_i$. Then each fraction can be interpreted as the marginal cost of reliability of deviating from the optimal path. Since each subsequent node on the optimal path has lower expected value of information, this result suggests that the marginal cost of deviation in terms of reliability must be higher for each node's ancestor where the expected value of information is also higher.

We define the cheapest neighbor path (CNP) from s_r to s_q as the simple path obtained by each node choosing its successor via its cheapest link (that connects to s_q). In a sense, this path reflects the route obtained when each node has only limited network state information (about neighbor costs and probabilities) and, in the absence of gradient information or route quality feedback, should merely minimize its local communication costs. The following proposition identifies when the CNP will coincide with the optimal path.

Proposition 37.3. *Given G and $P(s_i) = p \in (0, 1)$, for all i, the optimal RQR path is at least as reliable as the cheapest neighbor path. Furthermore, the CNP will be optimally reliable if*

$$\min\{c_k \backslash c_k^{\min}\} - c_k^{\min} > \quad v_r p^l(1 - p^{t-l})$$

where l is the length of the shortest path from s_r to s_q and t is the length of the CNP.

Proof. Consider an arbitrary node s_k which is k hops away from s_q on the CNP. Clearly, for the CNP to be optimal s_k should not get higher payoff by deviating to an alternative path. Also, we do not need to consider alternative paths that have lengths greater than k to s_q, since that would decrease benefits and

© 2005 by Chapman & Hall/CRC

the CNP already has the lowest cost edges. Let m be the path length along the CNP from s_r to s_k. For alternative paths of length $i = 1, \ldots, k-1$ from s_k to s_q to be infeasible, we need

$$c_i > c_o + v_r p^{m+i}(1 - p^{k-i})$$

where c_o is the edge cost along the CNP and c_i is the edge cost along alternative paths. By definition, for any node on the CNP, $m + i \geq l$. Also, at s_k we have $c_o = c^k_{\min}$, with c_i being at most $\min\{c^k \backslash c^k_{\min}\}$. Thus, when $\min\{c^k \backslash c^k_{\min}\} - c^k_{\min} > v_r p^l (1 - p^{t-l})$, the CNP will coincide with the optimal path.

The above proposition illustrates that the CNP does not have to be the most reliable in order to be optimal, it only needs to be sufficiently close. For networks in which some paths (edges) are overwhelmingly cheap compared with others, routing along CNPs may be reasonable. However, in networks where communication costs to neighbors are similar, routing based on local cost gradients is likely to be less reliable.

37.5 Path Weakness

We divide this section into two subsections. In the Section 37.5.1 we present our route evaluation metric and some theoretical results. Then Section 37.5.2 provides heuristics with low path weakness followed by simulation results about the quality of routes obtained using different routing algorithms. Throughout this section, we assume that there is a single source and destination pair. Thus, results are presented in terms of paths instead of trees.

37.5.1 Evaluation Metric

In an ideal sensor-centric network, optimal RQR paths are computed by individually rational sensors who maximize their own payoffs. On the other hand, traditional routing algorithms optimize using a single (end-to-end) distinguishing attribute, such as total cost or overall latency. From a sensor-centric perspective these approaches are inadequate and suboptimal, since they use a single network-wide criterion. How, then, do we compare different suboptimal paths? For example, one path may yield high payoffs for sensor i with low payoffs for sensor j, while the exact opposite situation may prevail on another path. Clearly, in a framework where rational, independent sensors maximize their own payoff subject to the overall network objective, we need a new metric for evaluating the quality of different paths from an individual sensor's point of view. We introduce a metric called path weakness, which captures the suboptimality of a node on the given path, i.e. how much a node would have gained by deviating from the current path to an optimal one. We believe this provides a new sensor-centric paradigm for evaluating the quality of routing in sensor networks.

We formally define our quality-of-routing (QoR) metric as follows. Let $\mathcal{P}$ be any given path from the source sensor s_r to the sink node s_q. Assume that the source contains information of value v_r and all other nodes have value $v_i = 0$. Consider any node s_i on $\mathcal{P}$ with ancestors $\{s_r, \ldots, s_{i-1}\}$. Let $\hat{\mathcal{P}}_{iq}$ be the optimal RQR path for routing information of value $\mathcal{V}_i = v_r \prod_{t=r}^{i} p_t$ (i.e. the expected value) to s_q from s_i in the subgraph $G\backslash\{s_r, \ldots, s_{i-1}\}$, assuming such a path exists. Thus, $\hat{\mathcal{P}}_{iq}$ represents the best that node s_i can do, given the links already established by nodes $s_r, \ldots, s_{i-1}$ *and* assuming optimal behavior from nodes s_i onward, downstream. Define $\Delta_i(\mathcal{P}) = \Pi_i(\hat{\mathcal{P}}_{iq}) - \Pi_i(\mathcal{P})$ as the payoff deviation for s_i under the given strategy profile (path) $\mathcal{P}$. A negative deviation represents the fact that s_i is benefiting more from this path (perhaps at the expense of some other sensor). Conversely, a positive deviation indicates s_i could have done better. We set $\Delta_i(\mathcal{P}) = v_r$ whenever $\Pi_i(\mathcal{P})$ is negative. This positive deviation from the optimal payoff is intended to represent the fact that s_i is participating in a path which is giving it negative payoffs, i.e. the communication cost on the edge out of s_i in $\mathcal{P}$ outweighs the benefits to s_i of participating in this route. Also note that it is possible that no optimal path from s_i exists, even if its payoff on $\mathcal{P}$ is positive. For example, all of s_i's neighbors might have very high communication costs

© 2005 by Chapman & Hall/CRC

and cannot participate in any optimal path, making s_i, in a sense, isolated. In such cases, we set $\Delta_i(\mathcal{P}) = -\Pi_i(\mathcal{P})$.

$\overline{\Delta}(\mathcal{P}) = \max_i \Delta_i(\mathcal{P})$ represents the payoff deviation at the node which is "worst-off" in $\mathcal{P}$. What can be said about this parameter for optimal and suboptimal paths?

Observation 37.2. *$0 < \overline{\Delta}(\mathcal{P}') \leq v_r$ for all nonoptimal paths $\mathcal{P}'$.*

However observe that $\Delta_i(\mathcal{P}')$, the weakness of individual nodes on suboptimal paths, can take both positive and negative values. On the other hand, $\overline{\Delta}(\mathcal{P}) = 0$ if and only if $\mathcal{P}$ is the Nash equilibrium path of the game. Thus, from a global point of view, $\overline{\Delta}(\mathcal{P})$ identifies the maximum degree to which a node on the path can gain by deviating. This allows us to rank the "vulnerability" of different paths, which embodies the idea that a path is only as good as its weakest node. We label this QoR measure the *path weakness*.

Note that the weakness metric can be similarly defined for data-aggregation trees. Given a sensor on any tree $\mathcal{T}$, its weakness can be calculated as its payoff deviation from the optimal tree that would have been obtained, given the expected value at that sensor along with the distribution of values in the remaining nodes in the graph. We can show that there exist networks not containing paths of bounded weakness. This result can be used to derive complexity bounds on finding paths with low weakness as follows:

Theorem 37.2. *There exists no polynomial-time algorithm to compute approximately optimal RQR paths of weakness less than $(v_r/3 - \epsilon)$ unless $P = NP$.*

As described by Kannan and Iyengar [7], the proof relies on constructing an instance of a sensor network in which the optimal RQR path is a well-known NP-complete problem instance (Hamiltonian path) and the next best path is separated from the optimal by a value of $v_r/3$.

Theorem 37.2 indicates the feasibility of finding approximately optimal RQR paths of bounded weakness. While this problem still remains open, Kannan and Iyengar [7] show that polynomial-time solutions for computing optimal RQR paths/trees for large classes of graphs exist.

Theorem 37.3. *Let G be any sensor network in which sensors are restricted to following a* **geographic routing** *regime. In other words, the strategy space of each sensor in the RQR game includes only those neighbors geographically nearer to the destination than itself. Then the optimal RQR path $\mathcal{P}$ can be computed in polynomial time in a distributed manner.*

37.5.2 Heuristics

We present some easy to compute heuristics, based on a team version of the RQR game (called TRQR), for finding approximate RQR paths. Simulation results verify that the TRQR heuristic has low path weakness and compares favorably with other standard routing algorithms.

The TRQR path can be interpreted game-theoretically as a "team" version of the RQR game in which all nodes on the path share the payoff of the worst-off node on it. Rather than selecting a neighbor to maximize their individual payoffs as in the original game, nodes in the TRQR model compromise by maximizing their least possible payoff. As before, each sensor's strategy is to select at most one next-neighbor (if the payoffs exceed its participation cost). Choices resulting in routing loops have zero payoffs. Formally, the payoffs to nodes in the network are defined as follows:

$$\Pi_i(l) = \begin{cases} v_r R(\mathcal{P}) - \max\limits_{(s_i, s_j) \in \mathcal{P}} c_{ij} & \text{if } s_i \in \mathcal{P} \\ 0 & \text{otherwise} \end{cases} \tag{37.1}$$

© 2005 by Chapman & Hall/CRC

where $R(\mathcal{P})$ is the reliability of path $\mathcal{P}$ from s_r (with value v_r) to s_q formed under strategy choice l. The Nash equilibrium of the TRQR game is the path from source to destination containing the node with the highest least cost-reliability trade-off over all paths. In case of multiple equilibria, the path with highest reliability is selected.

Note that the TRQR heuristic bears some similarity to the *bottleneck shortest path* problem, which minimizes the cost of the longest edge on the path from the source to the destination node. The optimal TRQR path can be interpreted as the bottleneck path to node s_q with the highest path reliability.

Formally, let $\overline{\mathcal{P}}_c$ represent the most reliable path from s_r to s_q that does not traverse any link exceeding cost c. Then the optimal TRQR path $\overline{\mathcal{P}}$ is given by

$$\overline{\mathcal{P}} = \arg\max_{c_i \in C} \left\{ v_r R(\overline{\mathcal{P}}_{c_i}) - c_i \right\} \tag{37.2}$$

for each distinct edge cost c_i in C. $\overline{\mathcal{P}}$ can be computed by repeatedly determining the most reliable path in the graph that is obtained by successively removing edges of decreasing distinct cost. In the worst case, m most reliable path calculations are made, where m is the number of distinct edge costs in the network.

37.6 Simulation Results

We simulate the performance of different routing algorithms to answer the following question: What are the quality of paths compared with that of the optimal RQR path? This allows us to identify the different ranges of node reliabilities and edge costs in which a particular algorithm performs better than the others.

The setup for our experiements is as follows. In every iteration, a random graph with 20 nodes and edge density of 30% is generated. The source and destination pair are randomly chosen and the value of the data at the source node is normalized to one. For each run, we choose a node survival probability, which is identical for all nodes. Communication costs over each edge are drawn randomly from a given parameter range in every iteration. For each set of node success probabilities and edge costs, we have presented results for 15 different source and destination pairs (we have verified that this is a representative sample). In each simulation run, for a particular source and destination pair, routing paths are generated by several algorithms and the corresponding path weakness (QoR) is calculated. The data have been used to construct graphs which are presented at the end of the paper. We have used the following algorithms: (1) the TRQR heuristic, (2) the most reliable path (MRP), (3) the cheapest next-node path (CNP), (4) the overall least-cost path (MCP), and (5) a genetic-algorithm-based heuristic (GA). (MRP) and (MCP) can be obtained using Djikstra's algorithm. The CNP is obtained by sequentially following the cheapest link out of each node that leads to the destination. The GA heuristic is based on the bicriteria shortest path solution provided by Gen and Cheng [18]. A path has been encoded according to the priority-based method. In this procedure, a set of n random numbers (n being the total number of sensor nodes) is generated so that the ith random number is the priority of the ith node. A path is sequentially constructed led by the highest priority feasible nodes, i.e. nodes which do not lead to a dead end or a cycle. The genetic operators used here are position-based crossover and swap mutation. A next generation is chosen by tournament method. We stop if the difference between the fitness values of the best paths of two adjacent generations is equal to zero.

37.6.1 Algorithm Analysis

Our simulation results are illustrated in Figures 37.1 and 37.2. Edge costs are low and chosen from a distribution such that every path is feasible (all possible node payoffs are positive).

In case I, we keep the node success probability fixed at 0.99 and vary the maximum edge cost from 0 to 0.05. The path weakness of the different algorithms ranges from 0 to 0.6, with TRQR having the

© 2005 by Chapman & Hall/CRC

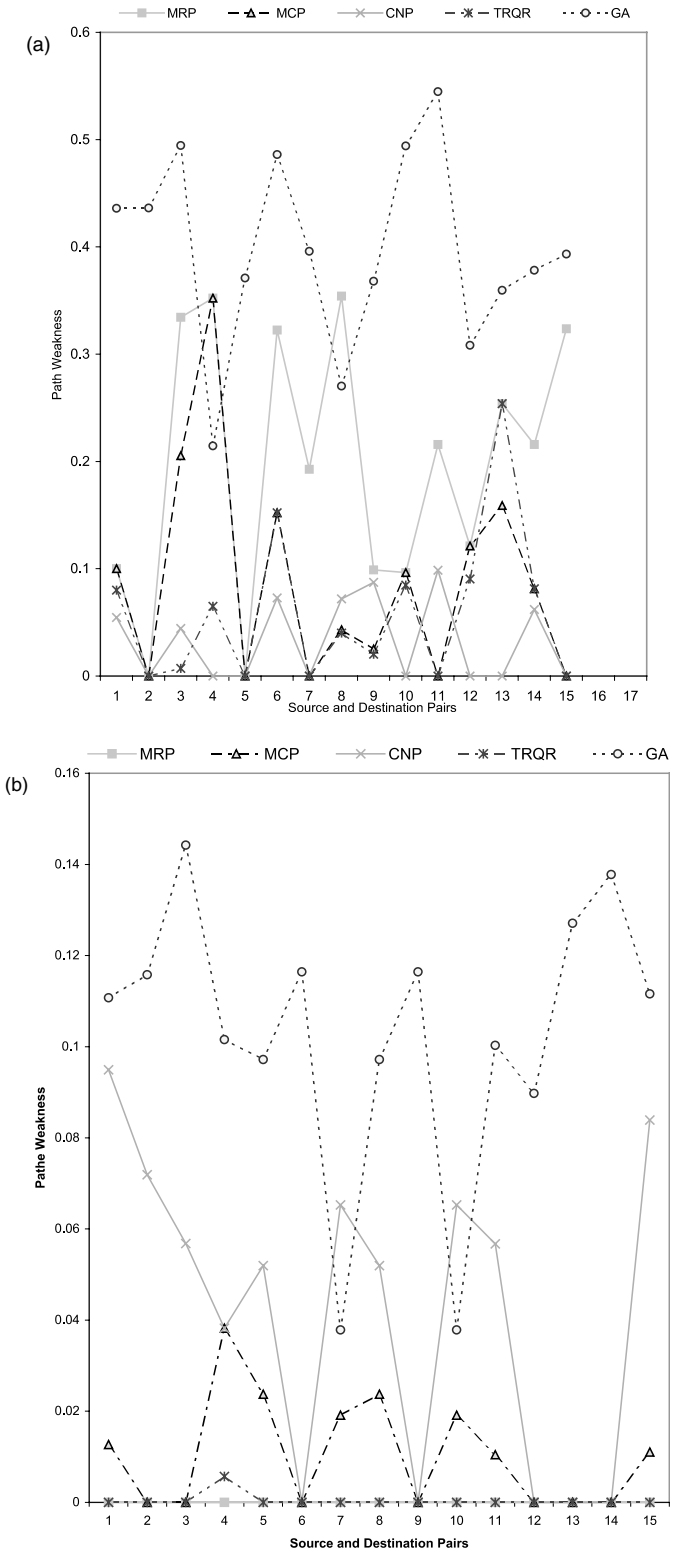
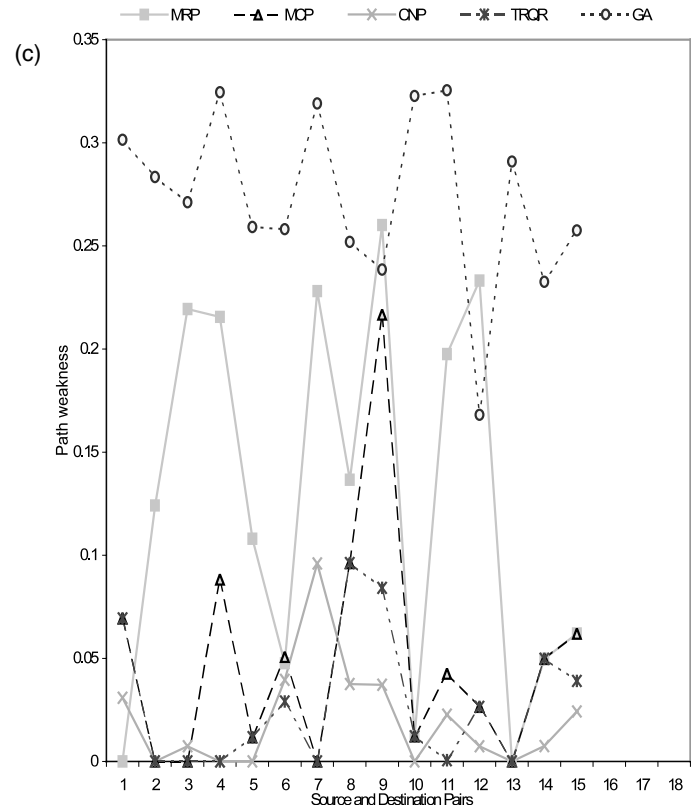


Figure 37.1. Simulation results. (a) Case I, $p = 0.99$, $c \leq 0.05$; (b) Case II, $p = 0.99$, $c \leq 0.01$; (c) Case III, $p = 0.992$, $c \leq 0.22$; (d) Case IV, $p = 0.998$, $c \leq 0.058$.

© 2005 by Chapman & Hall/CRC



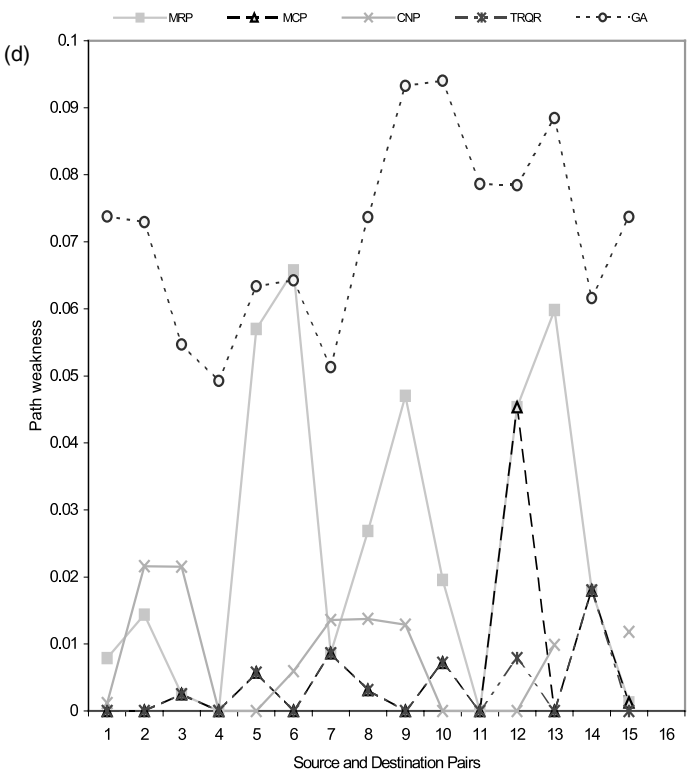


Figure 37.1. Continued.

© 2005 by Chapman & Hall/CRC

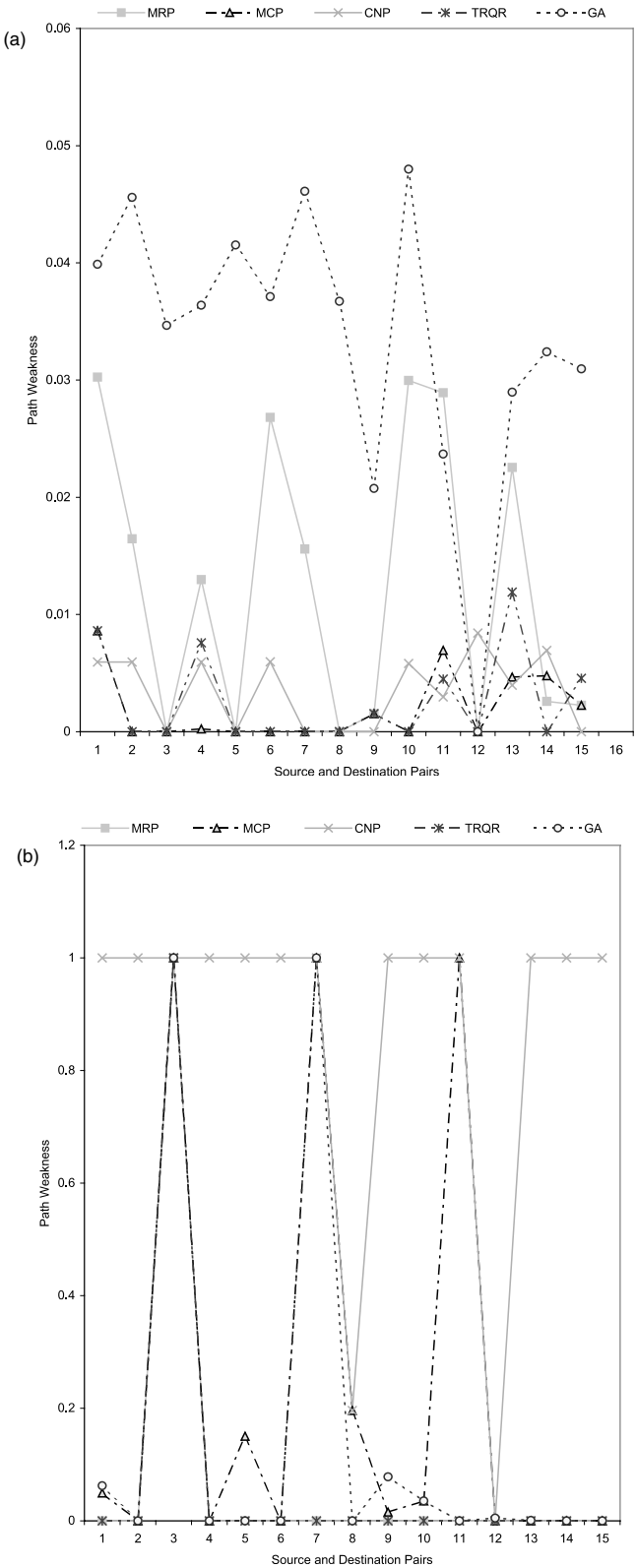


Figure 37.2. Simulation results. (a) Case V, $p = 0.999$, $c \leq 0.029$; (b) Case VI, $p = 0.5$, $c \leq 0.065$.

© 2005 by Chapman & Hall/CRC

lowest average weakness of 0.05. Since the cost range, and hence the cost differences, among various edges are not significantly large, all three cost-based algorithms (TRQR, MCP and CNP) have low path weakness, with TRQR having the lowest average weakness. However, the path weakness values of MRP suggest that, in this cost range, a path which relies solely on maximizing reliability cannot perform well.

In case II, node success probability is identical to case I, but edge costs are reduced to the 0–0.01 range. Consequently, the overall range of path weakness reduces to 0–0.14. Significant improvement takes place in the behavior of MRP, since most reliable paths also have low edge costs under this distribution. TRQR still has low path weakness.

For cases III, IV and V, we make the maximum edge cost a decreasing function of the node success probability. Then, we slowly increase node success probability to observe the impact. In case III, where the node success probability is 0.992 and the cost range is 0–0.227, the range of path weakness is quite high (0–0.35). When we raise the value of the success probability, the optimal paths can have longer lengths without sacrificing too much reliability. Therefore, CNP, which tends to have a longer length, has lower path weakness now. The TRQR heuristic, which trades off both the overall path reliability and the overall cost, performs as well as CNP, producing an average path weakness of 0.32. MRP again has higher path weakness due to the presence of a large number of comparable paths with high reliabilities. In case IV, the success probability is increased to 0.998 and the cost range is reduced to 0–0.058. This accounts not only for the relatively small range of path weakness (0–0.1), but also for the good performance of MCP, CNP, and TRQR. The congruence of TRQR and MCP is well explained by the significantly large difference between the success probability and the maximum edge cost. Case V is similar. In case VI we explore the consequences of restricting the likely optimal path length using one low node success probability (0.5) and maximum edge cost $(1/2)^4$. MRP, the shortest path, always coincides with the optimal path even though the success probability is quite low. So do TRQR and MCP. However, since the CNP usually has longer path lengths, it is quite weak in most cases.

When we compare all the graphs, we observe that, in networks with very high path reliabilities and low costs, TRQR and MCP have low weakness and outperform the other two algorithms. In general, MRP will be a good heuristic for obtaining good QoR paths only when path reliabilities are low. CNP provides good QoR when the success probability increases and the maximum edge cost decreases accordingly, but it is a bad choice for unreliable networks. Overall, the TRQR heuristic performs quite well in all cases, and it has low path weakness as it inherits the characteristics of MRP in unreliable networks and that of the cost-optimizing algorithms in higly reliable networks.

37.7 Conclusions

We have described a sensor-centric model of intelligent sensors using game theory. The problem of routing data in such a network is studied under the assumption that sensors are rational and act to maximize their own payoffs in the routing game. Further, nodes in our model are susceptible to failure and each node has to incur costs in routing data. To evaluate the contribution of individual nodes in the routing tree, the path weakness metric is developed. This individual-sensor-oriented evaluation criterion provides a new paradigm for examining the QoR paths. While the optimal routing problem turns out to be computationally hard, our experimental results show that standard path routing mechanisms, like MRP and MCP, usually find reasonably good paths. Our game-theoretically oriented algorithm, TRQR, compares favorably to the other standard routing algorithms.

Among the open issues worthy of investigation are the development of bounded, approximately optimal RQR paths/trees for general sensor networks[2] and RQR extensions using distributed and cooperative game models. Another interesting and open issue is the stability of the network under dynamic routing scenarios. Specifically, how stable is the Nash equilibrium of the RQR game in a dynamic environment where links and sensors fail periodically?

[2]Polynomial-time solutions for the optimal RQR and delay constrained paths/trees are presented by Kannan and Iyengar [7] for special classes of sensor graphs.

© 2005 by Chapman & Hall/CRC

Acknowledgments

The authors gratefully acknowledge the support of the DARPA SensIT program and AFRL under grant numbers F30602-01-1-0551 and F30602-02-1-0198 for the work described in this paper.

References

[1] Akyldiz, I.F. *et al.*, Wireless sensor networks: a survey, *Computer Networks*, 38(4), 393, 2002.
[2] Johnson, D.B. and Maltz, D.A., Dynamic source routing in *ad hoc* wireless sensor networks, *Mobile Computing*, Imielinski, T. and Korth, H., Eds., Kluwer Academic, Dordrecht, 1996.
[3] Perkins, C. and Royer, E., *Ad hoc* on demand distance vector routing, in *Proceedings of 2nd IEEE Workshop Mobile Computer Systems and Applications*, February 1999.
[4] Yu, Y. *et al.*, Geographical and energy aware routing: a recursive data dissemination protocol for wireless sensor networks, UCLA Computer Science Department Technical Report UCLA/CSD-TR-01-0023, May 2001.
[5] Brooks, R., Griffin, C., and Friedlander, D.S. Self-Organized distributed sensor network entity tracking, *International Journal of High Performance Computer Applications*, special issue on Sensor Networks, 16(3), 207–220, 2002.
[6] Chu, M. *et al.*, Scalable information-driven sensor querying and routing for *ad hoc* heterogeneous sensor networks, Xerox-PARC Technical Report P2001-10113, July 2001.
[7] Kannan, R. and Iyengar, S.S., Game theoretic models for length-energy-constrained routing in sensor networks, submitted to *IEEE Journal on Selected Areas in Communications*.
[8] Krishnamachari, B. *et al.*, Modeling data-centric routing in wireless sensor networks, in *Proceedings of INFOCOM 2002*, New York, June 2002.
[9] Rodoplu, V. and Meng, T.H., Minimum energy mobile wireless networks, *IEEE Journal on Selected Areas in Communications*, 17(8), 1333, 1999.
[10] Intanagonwiwat, C. *et al., Directed diffusion: a scalable and robust communication paradigm for sensor networks, in Proceedings of Sixth Annual International Conference on Mobile Computing and Networks (MobiCom 2000), Boston, MA, August 2000.*
[11] Sohrabi, K. *et al.*, Protocols for self-organization of a wireless sensor network, *IEEE Personal Communications*, October, 16, 2000.
[12] Shah, R.C. and Rabaey, J.M., Energy aware routing for low energy *ad hoc* sensor networks, in *Proceedings of IEEE Wireless Communications and Networking Conference (WCNC)*, Orlando, FL, March 2002.
[13] Wood, A. and Stankovic, J., Denial of service in sensor networks, *IEEE Computer*, October, 54, 2002,
[14] Zhou, L. and Haas, Z., Securing *ad hoc* networks, *IEEE Network*, 13(6), 24, 1999.
[15] Kannan, R. *et al.*, Sensor-centric reliable energy-constrained routing for wireless sensor networks, submitted to *Journal on Parallel and Distributed Computing*.
[16] Garey, M.R. and Johnson, D.S., *Computers and Intractability*, Freeman Publishing, New York, 1979.
[17] Fudenberg, D. and Tirole, J., *Game Theory*, MIT Press, 1991.
[18] Gen, M. and Cheng, R., *Genetic Algorithms and Engineering Optimization*, Wiley–Interscience, New York, 1999.
[19] Pottie, G., Hierarchical information processing in distributed sensor networks, in *Proceedings of International Symposium on Information Theory*, August 1998, 163.

© 2005 by Chapman & Hall/CRC

VI

Adaptive Tasking

38. Query Processing in Sensor Networks *R.R. Brooks* *751*
Introduction • Architecture for Query Processing in Sensor Networks • Sensor-Network-Specific Techniques and Optimizations • Experiments with Data Collection • Related Work • Concluding Remarks and Future Challenges • Acknowledgments

39. Autonomous Software Reconfiguration *R.R. Brooks* *773*
Problem Statement • Resource Constraints • Example Application Scenario • Distributed Dynamic Linking • Classifier Swapping • Dependability • Related Approaches • Summary • Acknowledgments and Disclaimer

40. Mobile Code Support *R.R. Brooks and T. Keiser* *787*
Problem Statement • Mobile-Code Models • Distributed Dynamic Linking • Daemon Implementation • Application Programming Interface • Related Work • Summary • Acknowledgments and Disclaimer

41. The Mobile-Agent Framework for Collaborative Processing in Sensor Networks *Hairong Qi, Yingyue Xu, and Teja Phani Kuruganti* *801*
Introduction • Mobile-Agent-Based Distributed Computing • The MAF • Application Examples • Summary

42. Distributed Services *Alvin S. Lim* *819*
Introduction • Purposes and Benefits of Distributed Services • Preview of Existing Distributed Services • Architecture of a Distributed Sensor System • Data-Centric Network Protocols • Distributed Services for Self-Organizing Sensor Applications • Application Systems • Conclusions • Acknowledgments

43. Adaptive Active Querying *Bhaskar Krishnamachari* *835*
Introduction • Active Queries as Random Walks • Active Queries with Direction • Conclusions

© 2005 by Chapman & Hall/CRC

In the wake of failures or degradation of embedded sensors within a network, there is a need for architectures, models, and algorithms that create failure resistant networks. By adapting to different kinds of circumstances, the distributed sensor networks (DSNs) could negate the ill effects of various kinds of failures. Different issues ranging from software adaptation for networks, implementation of mobile code daemons for adaptive reconfiguration, mobile-agent-based computing for collaborative processing, distributed services, and mechanisms used to implement active querying in sensor networks are discussed in this chapter.

Madden and Gehrke describe database management aspects of sensor networks. This includes query-processing issues. Queries can be defined at a higher level. The distributed database system is then responsible for assigning query components to proxy processes on the sensor nodes.

Brooks discusses software adaptation for networks of embedded processors. A general framework, known as distributed dynamic linking, is described in detail. He motivates the use of mobile code to autonomously reconfigure software on individual nodes.

Brooks and Keiser concentrate on the implementation of mobile code daemons for adaptive reconfiguration in distributed systems. They also provide background on mobile code paradigm. Further, they discuss their design of a system based on lightweight mobile code daemons. They also give an in-depth discussion of the indexing approach used for resource discovery in the mobile code, architecture, as well as the application-programming interface developed.

Qi *et al.* present the usage of mobile-agent-based computing paradigm for collaborative processing in sensor networks. They disucss the principles of mobile-agent-based computing as well as its fundamental differences from the client/server-based computing. Further, they also design and develop a mobile agent framework (MAF).

Lim discusses the purposes and benefits of distributed services, which are necessary for enabling sensor nodes to self-organize into impromptu networks that are incrementally extensible and dynamically adaptable to node failure and degradation, mobility and changes in task, and network requirements. The author further proposes an architecture for a self-organizing distributed sensor system, data-centric network protocols, distributed services for self-organizing sensor applications, application systems, etc.

Krishnamachari focuses on a discussion of different mechanisms that can be used to implement active querying in sensor networks. He begins by discussing the simple idea of random walk, and describes how it is used for different active querying techniques. He then discusses the possibility of sending active queries on predetermined trajectories; improvement of active query performance using reinforcement learning; and the use of geographic and senior information to direct the query.

In summary, this section discusses issues related to resistance to failures, and adapt to different possibilities in DSNs.

© 2005 by Chapman & Hall/CRC

38

Query Processing in Sensor Networks

Samuel Madden and Johannes Gehrke

38.1 Introduction

Recent advances in computing technology have led to the production of a new class of computing device: the wireless, battery-powered, smart sensor. Traditional sensors deployed throughout buildings, labs, and equipment are passive devices that simply modulate a voltage based on some environmental parameter. In contrast, these new sensors are active, fully fledged computers, capable not only of sampling real-world phenomena, but also of filtering, sharing, and combining those sensor readings with each other and nearby Internet-equipped endpoints.

As an example of a specific instance of a sensor network platform, consider the small sensor devices called *motes* developed at UC Berkeley. Current-generation motes are equipped with a 38.4 Kbit radio, an 8-bit address space, a 7 Mhz microprocessor, and a suite of sensors for measuring light, vibration, humidity, air pressure, magnetic field or gas or contaminate concentration. They are equipped with a small battery pack that provides sufficient energy for a few days of continuous operation, but they can be made to last for months or years if energy utilization is carefully managed. Motes run an operating system called TinyOS [1] that is especially suited to their capabilities. Networks of motes are usually deployed in an *ad hoc* fashion. These *ad hoc* networks differ from traditional networked environments in that motes in an *ad hoc* network can locate each other and route data without any prior knowledge or assumptions about the network topology. Figure 38.1 shows a current-generation mote in a small form factor with a weather sensor board.

Smart-sensor technology has enabled a broad range of ubiquitous computing applications: the low cost, small size, and untethered nature of these devices make it possible to sense information at previously unobtainable resolutions. Animal biologists can monitor the movements of hundreds of different animals simultaneously, receiving updates of location and ambient environmental conditions every few seconds. Vineyard owners can place sensors on every one of their plants, providing an exact picture of how various light and moisture levels vary in the *microclimates* around each vine. Supervisors of manufacturing plants, temperature-controlled storage warehouses, and computer server rooms can

© 2005 by Chapman & Hall/CRC



Figure 38.1. A Mica2Dot mote next to an AA battery. The top board is a weather-sensor board with light, temperature, and humidity sensors. The middle board contains the processor, radio, and nonvolatile flash. The bottom board and silver cylinder are the battery and battery connector.

monitor each piece of equipment, and automatically dispatch repair teams or shutdown problematic equipment in localized areas where temperature spikes or other faults occur.

Deployments such as those described above require months of design and engineering time, even for a skilled computer scientist. Some of this cost is hardware related and domain specific: for example, the appropriate choice of sensing hardware and device enclosure will vary dramatically if a network is designed for a forest canopy versus a sea floor. Aside from these domain-specific considerations, however, there is a substantial collection of software functionality common to each of these deployments: they all collect and periodically transmit information from some set of sensors, and they all need to carefully manage limited power and radio bandwidth to insure that essential information is collected and reported in a timely fashion.

To that end, the primary goal of our research is to design and implement an architecture upon which such data-collection applications can be built in dramatically less time. The key idea behind this architecture is that users specify the data they are interested in collecting through simple, *declarative* queries, just as in a database system, and that the infrastructure efficiently collects and processes the data within the sensor network. In contrast to traditional, embedded-C-based programming models, where each is treated as a separate computational unit, these queries are high-level statements of logical interests over an entire network, such as ''tell me the average temperature on the fourth floor of this building'' or ''tell me the location of the sensor with the least remaining battery capacity.'' The database system manages the details of data collection and processing (freeing the user from these concerns); in particular, it provides facilities for:

- Dissemination of queries into the sensor network.
- Identification of sensors which correspond to the logical names used in queries (e.g. ''sensors on the fourth floor'').
- Collection and processing of results from the network, over multiple radio hops and in a power efficient manner.
- Energy conservation.

© 2005 by Chapman & Hall/CRC

- Acquisition of sensor readings from a variety of low-level hardware interfaces.
- Storage and retrieval of collections of results in the network.
- Adaptation of communication topology and data rates to optimize network performance.

At Berkeley and Cornell, we have built several prototype sensor network query processors (SNQPs) which are instantiations of this architecture. The two systems, called Cougar [2] and TinyDB [3] respectively, run on a variety of different sensor platforms, including the Berkeley motes.

Aside from greatly simplifying the amount of work which users of sensor networks must do to prepare for a deployment of sensors, this query-processing-based approach to sensor management has the potential to offer dramatic improvements in energy efficiency — the typical measure of performance in sensor networks — of these data-collection applications. Again, this echos a well-known lesson from the relational database community: because declarative queries include no specification of *how* the required data are collected and processed, the system is free to explore many possible physical instantiations (plans) that have the same logical behavior as the user's query, and choose the one which is expected to offer the best performance. This process, termed *query optimization*, is central to the performance of our architecture.

In this chapter we describe our experiences designing the TinyDB and Cougar query processors, discussing the unusual challenges and novel features required of a high-level data-processing system in the world of sensor networks. We primarily discuss networks composed of homogeneous collections of Mica motes, though our work is general enough to be applicable outside of this regime. Indeed, initial versions of Cougar were implemented on nodes from Sensoria Corporation: the first generation was running on Windows CE and the second generation was running on an embedded version of Linux.

38.2 Architecture for Query Processing in Sensor Networks

Sensor networks provide a surprisingly challenging programming and computing environment: the devices are small and crash-prone; and the operating system that runs on them does not provide benefits like fault isolation to help mitigate such failures. Debugging is usually done via a few LEDs on the device. Programs are highly distributed, and must carefully manage energy and radio bandwidth while sharing information and processing with each other.

Because of limitations imposed by this impoverished computing environment, data-collection systems in sensor networks are required to support an unusual set of software requirements, such as:

- They must carefully manage resources, in particular power. Communication and sensing tend to dominate power consumption given the quantities of data and complexity of operations that are feasible on sensor networks. Furthermore, Moore's law suggests that the energy cost per CPU cycle will continue to fall [4] as transistors get smaller and use lower voltage, whereas fundamental physical limits and trends in battery technology suggest that the energy to transmit data via radio will continue to be expensive relative to the energy density of batteries. This has led us to focus much of our work on minimizing and optimizing for the energy costs of query processing in this environment.
- They have to be aware of and manage the transient nature of sensor networks: nodes come and go, signal strengths between devices vary dramatically as batteries run low and interference patterns change, but data collection should be interrupted as little as possible.
- They must be able to reduce and summarize data online while providing storage, logging, and auditing facilities for offline analysis. Transmitting all of the raw data out of the network in real time is often prohibitively expensive (in terms of energy) or impossible given data collection rates and limited radio bandwidth. Instead, small summaries, or *aggregates* (such as averages, moments, histograms, or statistical summaries) can be provided in real time. Many users, particularly scientists and the military, however, must eventually be able to collect and permanently store raw data, even if that data is not extracted from the network for several days or weeks.

© 2005 by Chapman & Hall/CRC

- They must provide an interface that is substantially simpler than the embedded-C-based programming model of TinyOS. While being simple, this interface must also allow users to collect desired information and process it in useful ways.
- Users must be given the tools to manage and understand the status of a network of sensors that have been deployed, and it must be it easy to add new nodes with new types of sensors and capabilities.

Note that each of these points represents a dissertation's worth of research, much of which still remains undone. Our goal in this chapter is to survey the current state of the art, describing at a high level the software and languages we have developed to address these challenges.

38.2.1 Architectural Overview

Figure 38.2 shows a simple block diagram of an architecture for query processing in sensor networks. The main two pieces of this architecture are:

- Server-side software that runs on the user's PC (the *base station*), which, in its most basic form, parses queries and delivers them into the network and collects results as they stream out of the network. In this chapter we will not discuss many of the details of server-side query processing; see Madden and co-workers [5,6] for more information.

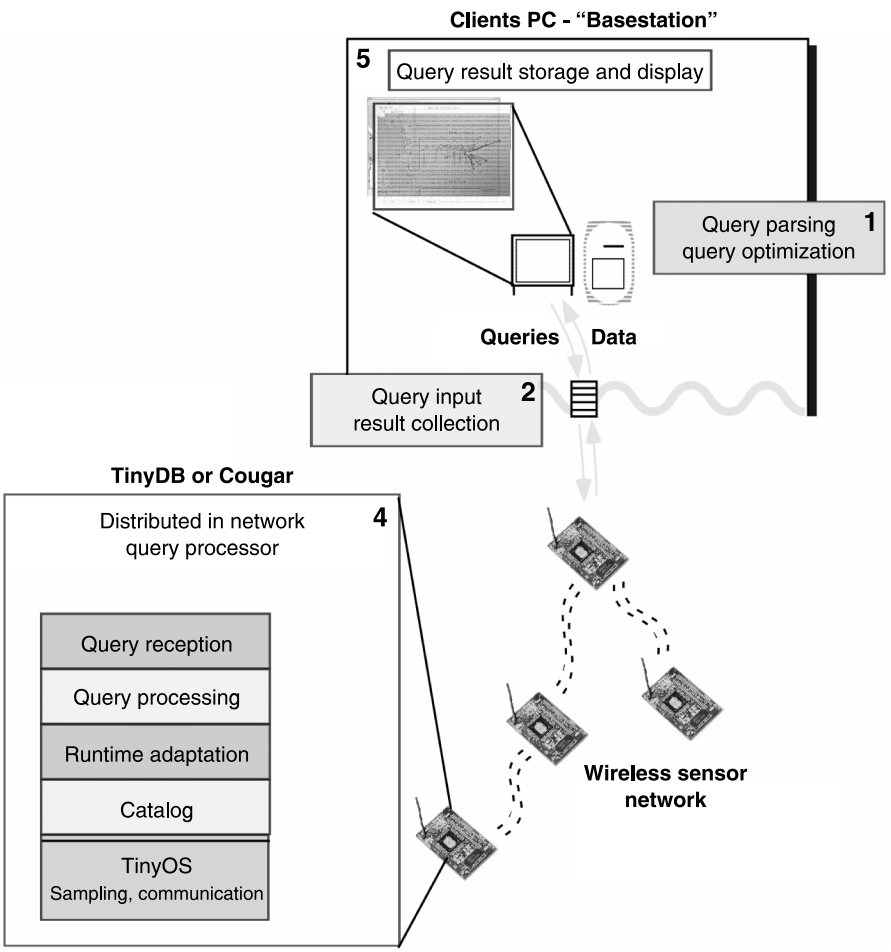


Figure 38.2. The architecture of an SNQP. Numbers indicate the sequence of steps involved in processing a query.

© 2005 by Chapman & Hall/CRC

- Sensor-side software that runs on the motes. As shown in the "Distributed In Network Query Processor" detail box on the left side of Figure 38.2, this software consists of a number of components built on top of TinyOS.

38.2.2 Introducing Queries and Query Optimization

In our architecture, queries are input at the server in a simple, SQL-like language which describes the data the user wishes to collect and ways in which they would like to combine, transform, and summarize it. The most significant way in which the variant of SQL we have developed differs from traditional SQL is that queries are *continuous* and *periodic*. That is, users register an interest in certain kinds of sensor reading (e.g. "temperatures from sensors on the fourth floor every 5 s") and the system streams these results out to the user. We call each period in which a result is produced an *epoch*. The *epoch duration*, or *sample period* of a query, refers to the amount of time between successive samples; for this example, the sample period would be 5 s. As we discuss various aspects of our system, we will show some examples of our language syntax and discuss its other features (new and in common with traditional SQL) in more detail.

Just as in a traditional database system, queries describe a *logical* set of data that the user is interested in, such as "the average temperature on the fourth floor," but they do not describe the actual algorithms and software modules, or *operators*, which the system must use to collect the answer set. Typically, there are a number of alternative *plans*, or choices, and orderings of operators for any given logical query. For example, to find the average temperature of the sensors on the fourth floor, the system might collect readings from every sensor, then filter that list for sensors on the fourth floor and compute their average, *or* it might request that only sensors on the fourth floor provide their temperature, and then take the average of all the values it collects. In a sensor network, we expect that the latter plan will always be a better choice, since it requires only sensors on the fourth floor to collect and report their temperature.

The process of selecting the best possible plan is called *query optimization*. At a very high level, query optimizers work by enumerating a set of possible plans, assigning a cost to each plan based on estimated costs of each of the operators, and choosing the plan of the lowest cost.

In a sensor network, this process of query optimization is done as much as possible on the server-side PC, since it can be quite computationally intensive. However, because the server may not have perfect state about the status of the sensor network, and because costs used to optimize a query initially may change over its lifetime, it is sometimes necessary to *adapt* running query plans once they have been sent into the network. We will see a few examples of this when we discuss query optimization in more detail.

38.2.3 Query Language

Queries in Cougar and TinyDB, as in SQL, consist of `SELECT-FROM-WHERE-GROUPBY-HAVING` blocks supporting selection, join, projection, aggregation, and grouping. The systems also include explicit support for windowing and subqueries (in TinyDB via *storage points*), and TinyDB also has explicit support for sampling. In queries, we view sensor data as a single virtual table with one column per sensor type. Tuples are appended to this table periodically, at well-defined intervals that are a parameter of the query. This period of time between each sample interval is the *epoch*, as described above. Epochs provide a convenient mechanism for structuring computation to minimize power consumption. Consider the query:

```
SELECT nodeid, light, temp
FROM sensors
SAMPLE PERIOD 1s FOR 10s
```

This query specifies that each sensor should report its own id, light, and temperature readings once per second for 10 s. The virtual table `sensors` contains one column for every attribute available in the

© 2005 by Chapman & Hall/CRC

catalog and one row for every possible instant in time. The term *virtual* means that these rows and columns are not actually materialized, i.e. only the attributes and rows referenced in active queries are actually generated.

The results of this query stream to the root of the network in an online fashion, via the multi-hop topology, where they may be logged or output to the user. The output consists of an ever-growing sequence of tuples, clustered into 1 s time intervals. Each tuple includes a time stamp corresponding to the time it was produced.

Note that the `sensors` table is (conceptually) an unbounded, continuous *data stream* of values; as is the case in other streaming and online systems, certain blocking operations (such as sort and symmetric join) are not allowed over such streams unless a bounded subset of the stream, or *window*, is specified. Windows in TinyDB are defined as fixed-size materialization points over the sensor streams. Such materialization points accumulate a small buffer of data that may be used in other queries. Cougar has a similar feature called *view nodes* that can store intermediate query results similar to materialized views in relational database systems; data are pushed from sensors to view nodes and then either pulled through interactive queries or periodically pushed to other view nodes or a base station. We show the TinyDB syntax here for concreteness:

Consider, as an example the following query:

```
CREATE
STORAGE POINT recentlight SIZE 8 seconds
AS (SELECT nodeid, light FROM sensors
SAMPLE PERIOD 1s)
```

This statement provides a shared, local (i.e. single-node) location to store a streaming view of recent data similar to materialization points in other streaming systems like Aurora or STREAM [7,8], or materialized views in conventional databases.

Joins are allowed between two storage points on the same node, or between a storage point and the `sensors` relation, in which case `sensors` is used as the outer relation in a nested-loops join. That is, when a `sensors` tuple arrives, it is joined with tuples in the storage point at its time of arrival. This is effectively a *landmark query* [9] common in streaming systems. Consider, as an example:

```
SELECT COUNT(*)
FROM sensors AS s, recentLight AS rl
WHERE rl.nodeid = s.nodeid
AND s.light < rl.light
SAMPLE PERIOD 10s
```

This query outputs a stream of counts indicating the number of recent light readings (from zero to eight samples in the past) that were brighter than the current reading.

TinyDB and Cougar also include support for grouped aggregation queries. Aggregation has the attractive property that it reduces the quantity of data that must be transmitted through the network, and thus can reduce energy consumption and reduce bandwidth usage by replacing more expensive communication operations with relatively cheaper computation operations, extending the lifetime of the sensor network significantly. TinyDB also includes a mechanism for user-defined aggregates and a metadata management system that supports optimizations over them.

Note that aggregation is a very powerful paradigm that has applicability that goes far beyond simple averaging. For example, the Cougar system has support for object tracking: nodes have a signal processing layer that generates signatures for objects that are in the vicinity of a sensor. Cougar implements a tracking operator as an aggregation over a region of sensor nodes, whose detections are aggregated into an estimation of a track containing the estimated speed and direction of an object. Overlap between regions ensures that an accurate track exists at all times.

© 2005 by Chapman & Hall/CRC

In addition to aggregates over values produced during the same sample interval (for example, as in the `COUNT` query above), users want to be able to perform temporal operations. For example, in a building monitoring system for conference rooms, users may detect occupancy by measuring maximum sound volume over time and reporting that volume periodically:

```
SELECT WINAVG(volume, 30s, 5s)
FROM sensors
SAMPLE PERIOD 1s
```

This query will report the average volume over the last 30 s once every 5 s, acquiring a sample once per second. This is an example of a *sliding-window* query common in many streaming systems [8,9].

When a query is issued in TinyDB or Cougar, it is assigned an identifier (id) that is returned to the issuer. This identifier can be used to explicitly stop a query via a "`STOP QUERY` id" command. Alternatively, queries can be limited to run for a specific time period via a `FOR` clause, or can include a stopping condition as a *triggering condition* or *event*; see our recent SIGMOD paper on acquisitional query processing [10] for more detail about these language constructs.

38.2.4 Query Dissemination and Result Collection

Once a query has been optimized, it is *disseminated* into the network. We discuss one basic communication primitive, a *routing tree*. A routing tree is rooted at either the base station or a storage point and it allows the root of the network to disseminate a query and to collect query results. This routing tree is formed by forwarding the query from every node in the network: the root initially transmits the query; all *child* nodes that hear it process it and forward it on to their children, and so on, until the entire network has heard about the query.

Each radio message contains a hop-count, or *level*, indicating the distance from the broadcaster to the root. To determine their own level, nodes pick a *parent* node that is (by definition) one level closer to the root than they are. This parent will be responsible for forwarding the node's (and its children's) query results to the base station. We note that it is possible to have several routing trees if nodes keep track of multiple parents. This can be used to support several simultaneous queries with different roots. This type of communication topology is common within the sensor network community and is known as *tree-based routing.*

Figure 38.3 shows an example sensor network topology and routing tree. Solid arrows indicate parent nodes and dotted lines indicate nodes that can hear each other but do no use each other for routing. In general, a node may have several possible choices of parent; a simple approach is to chose the parent to be the ancestor node with the highest level. In practice, it turns out that making a proper choice of parent is quite important in terms of communication and data collection efficiency. Unfortunately, the details of the best known techniques for doing this are quite complicated and outside the scope of our discussion in this chapter. For a more complete discussion of these and other issues, see Chu *et al.* [11].

Once a routing tree has been constructed, each node has a connection to the root of the tree which is just a few radio hops long. We can then use this same tree to collect data from sensors by having them forward query results up this path.

Note that simple routing structures such as routing trees are very suitable for our scenario: SNQPs impose communication workloads on the multi-hop communication network that are very different from traditional *ad hoc* networks with mobile nodes. Since the sensor network is programmed only through queries, there are very regular communication patterns, mainly consisting of the collection of sensor readings from a region at a single node or the base station.

Note that other types of routing structure beyond routing trees are necessary if the query workload has more than a few destinations, since the overlay of several routing trees neglects any sharing between several trees and leads to performance decay. The discussion of such routing algorithms is beyond the scope of this chapter, but we have begun to explore such issues in our research.

© 2005 by Chapman & Hall/CRC

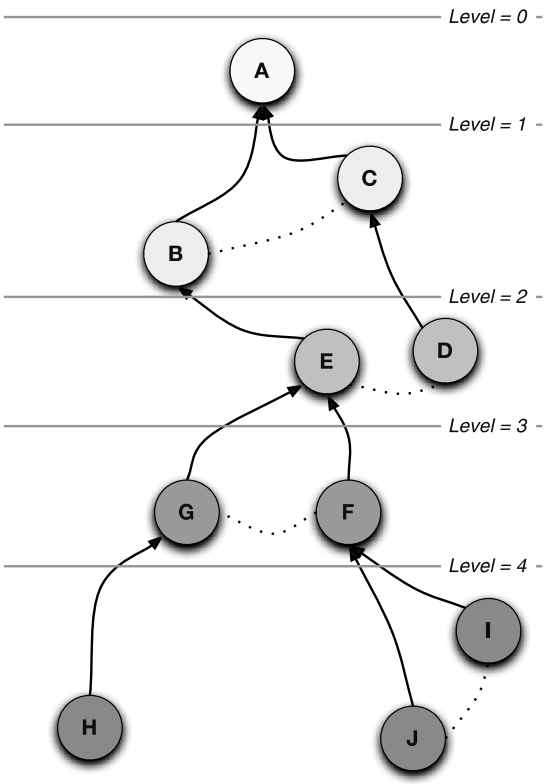


Figure 38.3. A sensor network topology, with routing tree overlay.

38.2.5 Query Processing

Once a query has been disseminated, each node begins processing it. Processing is a simple loop: once per epoch, readings, or *samples*, are acquired from sensors corresponding to the fields, or *attributes*, referenced in the query. This acquisition is done by a special *acquisition operator*. This set of readings, or *tuple*, is routed through the query plan built in the optimization phase. The plan consists of a number of operators that are applied in a fixed order; each operator may pass the tuple on to the next operator, reject it, or combine it with one or more other tuples. Any tuple that successfully passes the plan is transmitted up the routing tree to the node's parent, which may in turn forward the result on or may combine it with its own data or data collected from its other children. Table 38.1 describes some common query processing operators which are used in SNQPs.

The acquisition operator uses a *catalog* of available attributes to map names referenced in queries into low-level operating system functions that can be invoked to provide their values. This catalog abstraction allows sophisticated users to extend the sensor network with new kinds of sensor, and also provides support for sensors that are accessed via different software interfaces. For example, in the TinyDB system, users can run queries over sensor attributes like light and temperature, but they can also query attributes that reflect the state of the device or operating system, such as the free RAM in the dynamic memory allocator. Table 38.2 shows a list of some of the sensor and system attributes that are available on current-generation sensors. This table includes energy per sample as an example of additional catalog metadata that can be used in query optimization.

Figure 38.4 illustrates query processing for the simple aggregate query "Tell me the average temperature on the fourth floor once every 5 s." Here, the query plan running on every node contains three operators: an acquisition operator, a select operator that checks to see if the value of the `floor` attribute is equal to 4, and an aggregate operator that computes the average of the `temperature`

© 2005 by Chapman & Hall/CRC

Table 38.1. Common sensor network query processing operators

Operator	Description
`data acquisition`	Acquire a reading (field) from a sensor or an internal device attribute. Example attributes are light sensor readings or free RAM in the dynamic heap
`select`	Reject readings that do not satisfy a particular Boolean predicate. For example, the predicate `temp` > 80°F rejects readings under > 80°F
`aggregate`	Combines readings together according to an *aggregation function*. For example, `AVG(light)` computes the average light value over each mote
`join`	Concatenate two readings when some *join predicate* is satisfied. An example predicate might be `mat-point.light > sensors.light`, which joins (concatenates) all of the historical tuples in mat-point with current sensor readings for any pair of tuples where the current light value, exceeds the historical value

Table 38.2. Some sensors available for Mica motes and their power requirements

Sensor	Notes	Energy Per Sample @ 3 V (mJ)
Solar Radiation [12]	Amount of radiation in 400 to 700 nm range that allows plants to photosynthesize	0.525
Barometric Pressure [13]	Air pressure, in millibars	0.003
Humidity [14]	Relative humidity	0.5
Passive Infrared [15]	Temperature of an overhead surface	0.0056
Ambient Temp [15]		0.0056
Accelerometer [16]	Measure movement and vibration	0.0048
(Passive) Thermistor [17]	Uncalibrated temperature, low cost	0.00009

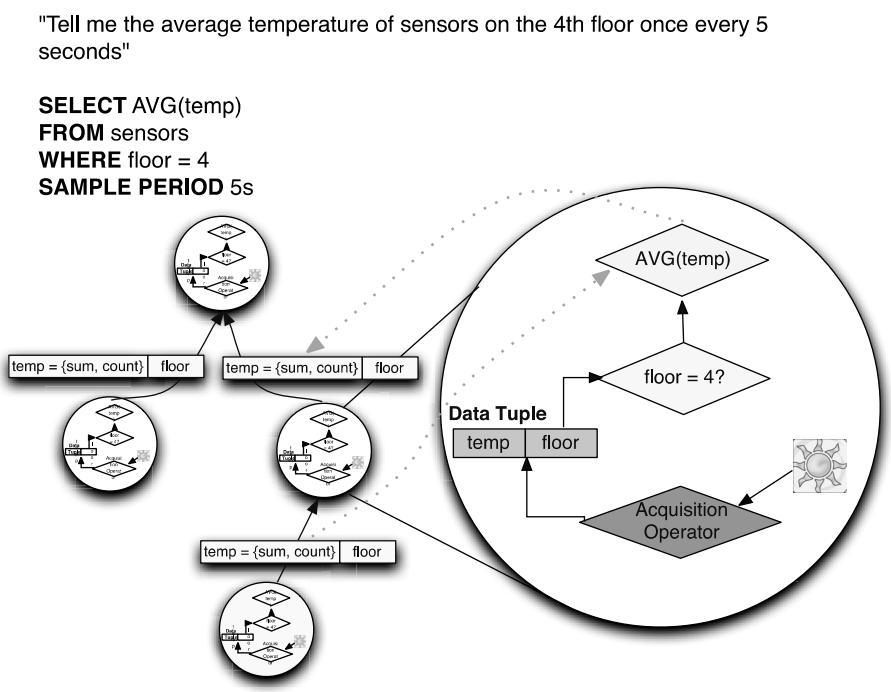


Figure 38.4. A sensor network executing a simple aggregate query.

© 2005 by Chapman & Hall/CRC

attribute from the local mote and the average temperature values of any that of mote's descendents (which happen to be on the fourth floor.) Each sensor applies this plan once per epoch, and the stream of data produced at the root node is the answer to the query. Note that we represent the partial computation of averages as `{sum, count}` pairs which are merged at each intermediate node in the query plan to compute a running average as data flow up the tree.

To make this scheme work, there are a number of implementation details that must be resolved: sensors must wait to hear from their children before reporting their own averages, and average records must be represented in such a way that they can be combined as they flow up the tree (in this case, as a sum and a count, rather than a simple average).

38.3 Sensor-Network-Specific Techniques and Optimizations

Now that we have a basic outline of how query processing in sensor networks functions, we devote the rest of the chapter to the unusual kinds of optimizations and query processing techniques which arise in the context of SNQPs.

38.3.1 Lifetime

In lieu of an explicit `SAMPLE PERIOD` clause, we allow users to specify a specific query lifetime via a `QUERY LIFETIME <x>` clause, where `<x>` is a duration in days, weeks, or months. Specifying lifetime is a much more intuitive way for users to reason about power consumption. Especially in environmental monitoring scenarios, scientific users may not be particularly concerned with small adjustments to the sample rate, nor do they understand how such adjustments influence power consumption. Such users, however, are very concerned with the lifetime of the network executing the queries.

Consider the following query:

```
SELECT nodeid, accel
FROM sensors
LIFETIME 30 days
```

This query specifies that the network should run for at least 30 days, sampling light and acceleration sensors at a rate that is as quick as possible and still satisfies this goal.

To satisfy a lifetime clause, the SNQP applies *lifetime estimation*. The goal of lifetime estimation is to compute sampling and transmission rates given a number of joules of energy remaining (which can *usually* be estimated from the battery voltage on the mote) and a specific query or set of queries to run. Note that, just as with query optimization, lifetime estimation can be done when a query is initially issued at the PC, or may be applied periodically, within the network as the query runs. We have currently implemented the former approach in TinyDB, but the latter approach will be more effective, especially in a network with lots of nodes communicating in unpredictable ways.

To illustrate the effectiveness of this simple estimation, we inserted a lifetime-based query (`SELECT voltage, light FROM sensors LIFETIME x`) into a sensor (with a fresh pair of AA batteries) and asked it to run for 24 weeks, which resulted in a sample rate of 15.2 s per sample. We measured the remaining voltage on the device nine times over 12 days. The first two readings were outside the range of the voltage detector on the mote (e.g. they read "1024," the maximum value) so are not shown. Based on experiments with our test mote connected to a power supply, we expect it to stop functioning when its voltage reaches 350. Figure 38.5 shows the measured lifetime at each point in time, with a linear fit of the data, versus the "expected voltage," which was computed using a simple cost model. The resulting linear fit of voltage is quite close to the expected voltage. The linear fit reaches $V = 350$ about 5 days after the expected voltage line.

Lifetime estimation is a simple example of an optimization technique which the sensor network can apply to provide users with a more useful, expressive way of interacting with a network of sensors.

© 2005 by Chapman & Hall/CRC

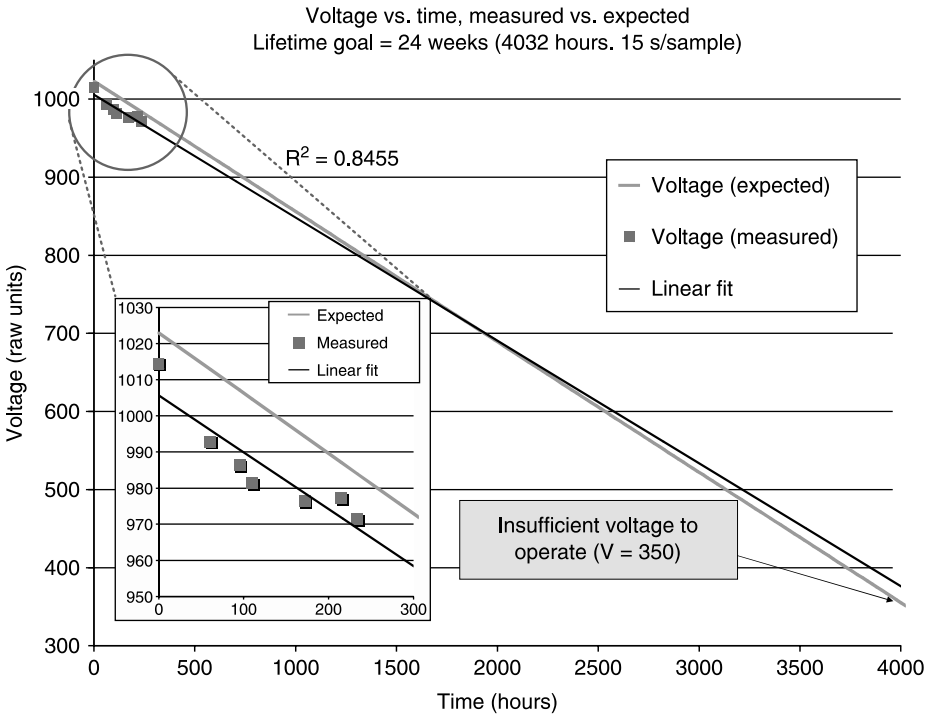


Figure 38.5. Predicted versus actual lifetime for a requested lifetime of 24 weeks (168 days).

38.3.2 Pushing Computation

Among the most general techniques for query optimization in sensor network database systems is the notion of *pushing computation*, or moving processing into the network, towards the origin of the data being processed. We introduce two forms of this technique for aggregate queries.

A query plan for a simple aggregate query can be divided into two components. Since queries require data from spatially distributed sensors, we need to deliver records from a set of distributed nodes to a central destination node for aggregation by setting up suitable communication structures for delivery of sensor records within the network. We call this part of a query plan its *communication component*. In addition, the query plan has a *computation component* that computes the aggregate at the network root and potentially computes already partial aggregates at intermediate nodes. We describe two simple schemes for pushing computation; more sophisticated push-based approaches are possible [18,19].

38.3.2.1 Partial Aggregation

For aggregates that can be incrementally maintained in constant space (or, using terminology from the database literature, for *distributive* and *algebraic* aggregate operators [20]), we push partial computation of the aggregate from the root node down to intermediate nodes. Each intermediate sensor node will compute partial results that contain sufficient statistics to compute the final result. `AVERAGE` is an example of an aggregate that has constant intermediate state and can be distributed in this way; the example given in Figure 38.3 illustrates the concept of pushing partial aggregation into the network.

38.3.2.2 Packet Merging

Since it is much more expensive to send multiple smaller packets instead of one larger packet (considering the cost of reserving the channel and the payload of packet headers), we can *merge* several records into a larger packet, and only pay the packet overhead once per group of records. For exact query answers with aggregate operators that do not have a compact incremental state representation like

© 2005 by Chapman & Hall/CRC

the Median (these are called *holistic aggregates*), packet merging is the only way to reduce the number of bytes transmitted.

38.3.3 Cross-Layer Interactions

In the previous section we saw that we can optimize aggregate operators through in-network aggregation, such as packet merging and partial aggregation at internal nodes. These techniques require internal nodes to *intercept* data packets passing through them to perform packet merging or partial aggregation. However, with the traditional "send and receive" interfaces of the network layer, only the root of the routing tree will receive the data packets. The network layer on an internal node will automatically forward the packets to the next hop towards the destination, and the upper layers will be unaware of data packets traveling through the node. Thus, a node needs some functionality to "intercept" packets that are routed through it, and the SNQP needs a way to communicate to the network layer when it wants to intercept packets that are destined for another node.

One possible way to implement this interception is through network filters, which is the approach taken in Cougar. With filters, the network layer will first pass a packet through a set of registered functions that can modify (and possibly even delete) it. In the query layer, if a node n is scheduled to aggregate data from all children nodes, then it can intercept all data packets received from it's children and cache the aggregated result. At a specific time, n will then generate a new data packet representing the incremental aggregation of it and its children's data and send it towards the root of the network. All this happens completely transparently to the network layer.

Another possibility to implement this interception is to collapse the network stack and to merge the routing layer with the application layer, which is the approach taken in TinyDB. In this case, the application has complete control over the routing layer, and each packet that is routed through a node is handled by the application-level routing layer.

Both of these approaches are instances of *cross-layer interactions*. In order to preserve resources, we believe that future generations of sensor networks will take an integrated approach toward the design of the system architecture, which crosscuts the data management and communication (routing and MAC) layers. We can distinguish two approaches: a *top-down* and a *bottom-up* approach. In the top-down paradigm, we design and adapt communication protocols and their interfaces to the particular communication needs of the SNQP. In the bottom-up approach, we consider the task of adapting query processing techniques to a given routing policy. Cross-layer interactions are a very fertile area of sensor network research, and TinyDB and Cougar have only made preliminary steps in this direction.

38.3.4 Acquisitional Query Processing

At first blush, it may seem as though query processing in sensor networks is simply a power-constrained version of traditional query processing: given some set of data, the goal of sensor network query processing is to process that data as energy efficiently as possible. Push-down strategies, such as those discussed above, are similar to push-down techniques from distributed query processing that emphasize moving queries to data.

There is, however, another fundamental difference between systems like sensor networks and traditional database systems, that has to do with the role of data acquisition in query processing. In this *acquisitional query processing* (ACQP), the focus is on the significant new query processing opportunity that arises in sensor networks: the fact that smart sensors have control over where, when, and how often data are physically acquired (i.e. sampled) and delivered to query processing operators. By focusing on the locations and costs of acquiring data, it is possible to reduce power consumption significantly compared with traditional passive systems that assume the *a priori* existence of data. Acquisitional issues arise at all levels of query processing: in query optimization, due to the significant costs of sampling sensors; in query dissemination, due to the physical collocation of sampling and processing; and in query execution, where choices of when to sample and which samples to process are made.

© 2005 by Chapman & Hall/CRC

We have designed and implemented ACQP features in TinyDB. While TinyDB has many of the features of a traditional query processor (e.g. the ability to select, join, project, and aggregate data), it also incorporates a number of other features designed to minimize power consumption via acquisitional techniques. These techniques, taken in aggregate, can lead to orders of magnitude improvement in power consumption *and* increased accuracy of query results over systems that do not actively control when and where data are collected.

One of the fundamental techniques derived from ACQP has to do with when sensor readings are acquired during the processing of a query. Because the cost of sampling sensors can be quite high — a significant fraction of the total processing cost for expensive, calibrated, digital sensors like the Honeywell Magnetometer [21] used in current-generation Mica Motes — it may be useful to postpone the acquisition of sensor readings until it is absolutely clear those readings will be needed.

As an example of a situation where postponing acquisition in this manner can be useful, consider the query below, noting that the cost to acquire a sample from the magnetometer [21] (the `mag attribute`) is several orders of magnitude greater than the cost to acquire a sample from a photo-resistor-based light sensor [17] on current-generation Mica hardware, i.e. 0.25 mW versus 0.09 μW:

```
SELECT WINMAX(light,8s,8s)
FROM sensors
WHERE mag > x
SAMPLE PERIOD 0.1s
```

In this query, the maximum of every eight seconds' worth of light readings will be computed, but only light readings from sensors whose magentometers read greater than x will be considered. Interestingly, it turns out that, unless the x predicate is *very* selective, it will be cheaper to evaluate this query by checking to see if each new `light` reading is greater than the previous reading and then applying the selection predicate over `mag`, rather than first sampling `mag`. This sort of reordering, which we call *exemplary aggregate push-down*, can be applied to any exemplary aggregate (e.g. `MIN`, `MAX`), and is general (i.e. applies outside the context of sensor networks). Furthermore, it can reduce the number of magnetometer samples that must be acquired by up to a factor of 80, which corresponds to a total energy saving of about 20 mJ, i.e. a power saving of 2.5 mW (recall that 1 mW = 1 mJ/s). This is roughly half the power required to run the processor.

Thus, properly choosing the order in which data are acquired can dramatically reduce energy consumption and is an example of a novel kind of optimization that does not arise in other query processing environments.

38.4 Experiments with Data Collection

We have done a number of studies of the performance and behavior of our SNQP implementations, both in simulation to demonstrate the potential of our algorithms and approaches and in real-world environments to observe their overall effectiveness.

38.4.1 Berkeley Botanical Garden Deployment

During June and July of 2003, we began a deployment of the TinyDB software in the Berkeley Botanical Garden, located on Centennial Road in Berkeley, just east of the main UC Berkeley Campus. The purpose of this deployment was to monitor the environmental conditions in and around Coastal Redwood trees (the *microclimate*) in the garden's redwood grove.

This grove consists of several hundred new-growth redwoods. Botanists at UC Berkeley [22] are actively studying these microclimates, with a particular interest in the role that the trees have in regulating and controlling their environment, especially the ways they affect the humidity and temperature of the forest floor on warm, sunny days.

© 2005 by Chapman & Hall/CRC

The initial sensor deployment in the garden consists of 11 Mica2 sensors on a single 36 m redwood tree, each of which is equipped with a weather board that provides light, temperature, humidity, solar radiation, photosynthetically active radiation, and air pressure readings. The sensors are placed in clusters at different altitudes throughout the tree. The processor and battery are placed in a water-tight PVC enclosure, with the sensors exposed on the outside of the enclosure. A loose-fitting hood covers the bottom of the sensor to protect the humidity and light sensors from rain. The light and radiation sensors on the top of the assembly are sealed against moisture, and thus remain exposed. Figure 38.1 shows an example of such a mote outside of its PVC package.

Sensors on the tree run a simple selection query which retrieves a full set of sensor readings every 10 min and sends them towards the base station, which is attached to an antenna on the roof of a nearby field station, about 150 ft from the tree. The field station is connected to the Internet; so, from there, results are easily logged into a PostgreSQL database for analysis and observation. The sensors have been running continuously for about 3 weeks. We expect the sensors to function for about 40 days, and plan eventually to grow the deployment to approximately 100 nodes over five trees.

Figure 38.6 shows data from five of the sensors collected during the second week of July, 2003. Sensor 101 was at a height of 10 m, sensor 104 at 20 m, sensor 109 at 30 m, sensor 110 at 33 m, and sensor 111 at 34 m. Sensors 110 and 111 we fairly exposed, while the other sensors remained shaded in the forest canopy.

The periodic bumps in the graph correspond to daytime readings; at night, the temperature drops significantly and humidity becomes very high as fog rolls in. Notice that 7/7 was a cool day, below 18°C and likely overcast as many summer days in Berkeley are. On days like this, all of the sensors record approximately the same temperature and humidity. Later in the week, however, it was much warmer, climbing as high as 28°C at the top of the tree. Note at these times, however, it can be as many as 10°C cooler at the bottom of the tree, with a 30% higher humidity. This observation should be familiar to anyone who has ever been walking in a redwood forest and felt the cool dampness of the forest floor below the trees.

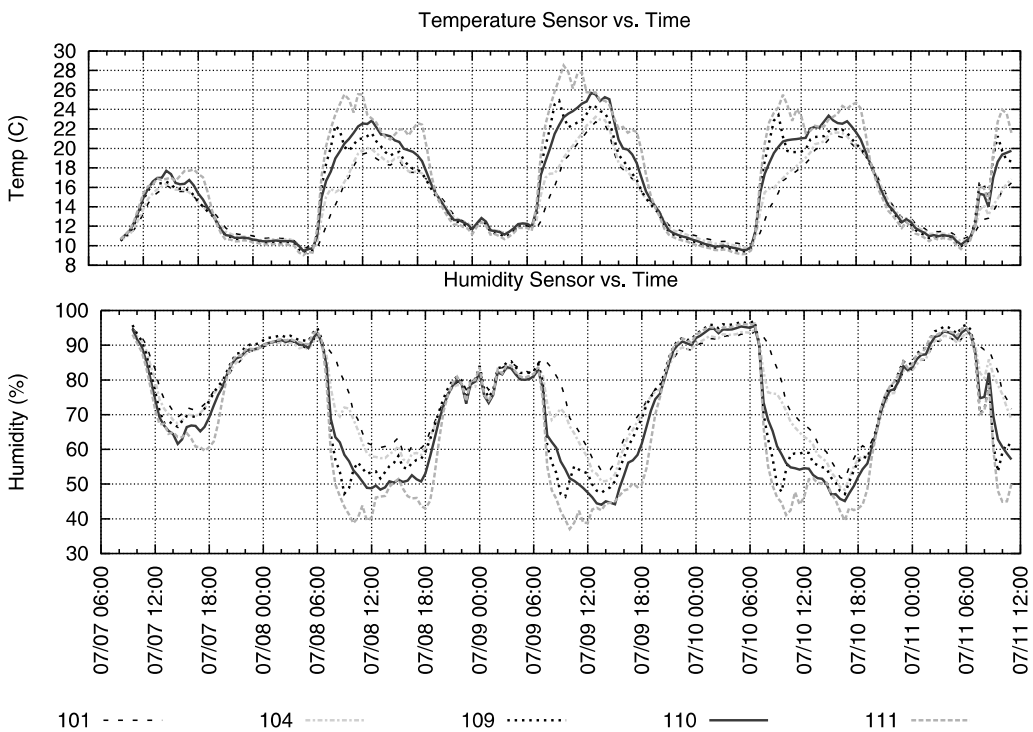


Figure 38.6. Humidity and temperature readings from five sensors in the Berkeley Botanical Garden.

© 2005 by Chapman & Hall/CRC

Although this is a fairly basic deployment, running only a simple query, we were able to program the sensors to begin data collection in just a few minutes. By far and away the most time-consuming aspects of the deployment involved the packaging of the devices, obtaining access to the various spaces, and climbing the tree to place the sensors.

In future versions of this deployment, we hope to move to an approach where we log all results to the EEPROM of the devices and then just transmit summaries of that data out of the network, only dumping the EEPROM on demand or during periods where there is no other interesting activity (e.g. at night.) Once scientists believe that our hardware and software function correctly, we believe they will be more likely to accept this style of approach.

Another interesting future direction related to this deployment has to do with tracking correlations between sensors and using those correlations to improve the efficiency of query processing. Note that, in Figure 38.6 the temperature and humidity are highly correlated — knowing the humidity and sensor number allows one to predict the temperature to within a few degrees Celsius. This observation suggests an interesting query optimization possibility: for queries that contain predicates over temperature, one might evaluate them instead by looking at humidity. If a humidity sample is needed for other purposes, or the energy costs of acquiring a humidity sample are low, this could be an energy-saving alternative.

38.4.2 Simulation Experiments

We also performed extensive simulation studies of our approach to show that it works well in a controlled environment — often, simulation is the only way to get repeatable results out of noisy, lossy sensor networks. For Cougar, we have a prototype of our query processing layer which runs in the ns-2 network simulator [23]. Ns-2 is a discrete event simulator targeted at simulating network protocols to highest fidelity. Owing to the strong interaction between the network layer and our proposed query layer, we simulate the network layer to a high degree of precision, including collisions at the MAC layer, and detailed energy models developed by the networking community. In our experiments, we used IEEE 802.11 as the MAC layer [24], setting the communication range of each sensor to 50 m and assuming bi-directional links; this is the setup used in most other papers on wireless routing protocols and sensor networks in the networking community [25]. In our energy model the receive power dissipation is 395 mW, and the transmit power dissipation is 660 mW [25]; the radio was turned off outside of its "slot" in the routing tree. Sensors are randomly distributed in square region of increasing size, keeping the average sensor node density constant at eight sensors per 100 m $\times$ 100 m. The root node is located in the upper left corner; we ran a simple query that computes the average over all sensors.

We compared three simple approaches:

- *Direct delivery*. This is the simplest scheme. Each source sensor node sends a data packet consisting of a record towards the leader, and the multi-hop *ad hoc* routing protocol will deliver the packet to the leader. Computation will only happen at the leader after all the records have been received.
- *Packet merging.* In wireless communication, it is much more expensive to send multiple smaller packets instead of one larger packet, considering the cost of reserving the channel and the payload of packet headers. Since the size of a sensor record is usually small and many sensor nodes in a small region may send packets simultaneously to process the answer for a round of a query, we can *merge* several records into a larger packet and only pay the packet overhead once per group of records. For exact query answers with holistic aggregate operators like Median, packet merging is the only way to reduce the number of bytes transmitted [26].
- *Partial aggregation.* For distributive and algebraic aggregate operators [26], we can incrementally maintain the aggregate in constant space, and thus push partial computation of the aggregate from the leader node to intermediate nodes. Each intermediate sensor node will compute partial results that contain sufficient statistics to compute the final result.

© 2005 by Chapman & Hall/CRC

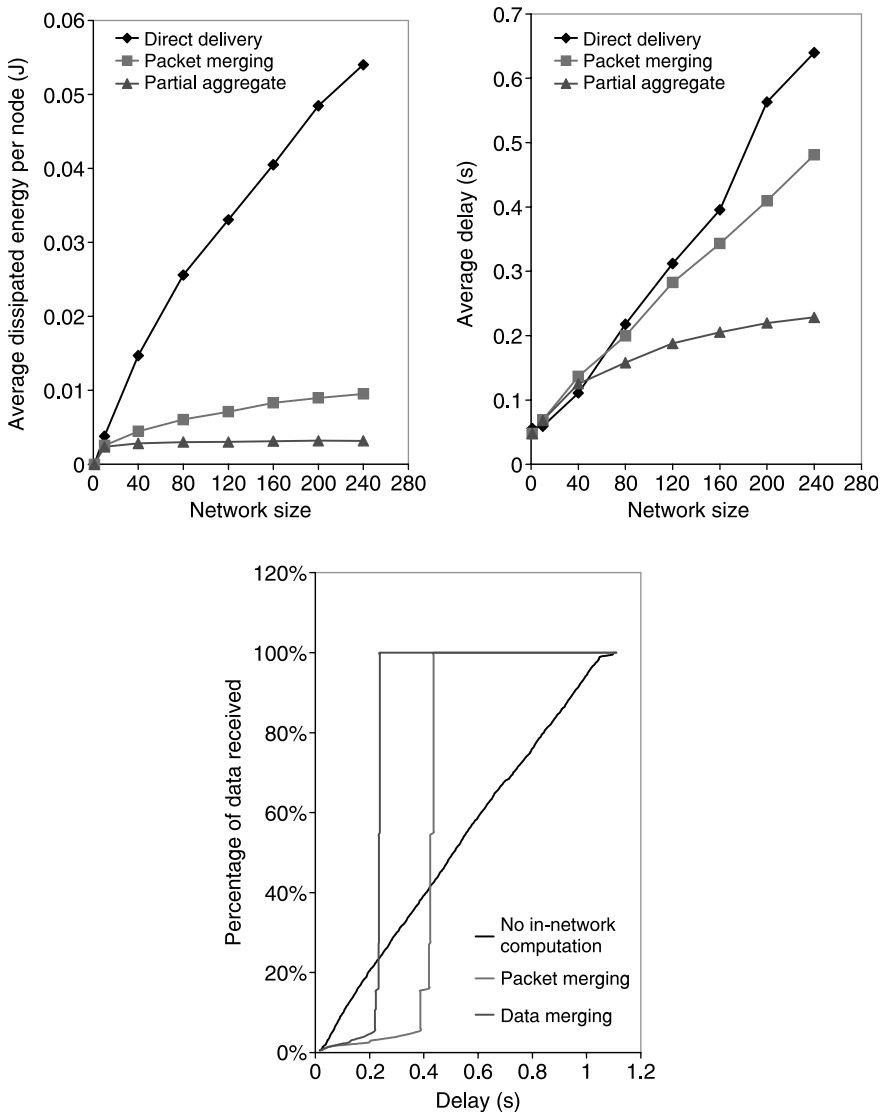


Figure 38.7. Simulation results comparing different approaches for answering an aggregate query.

The leftmost graph in Figure 38.7 illustrates the benefit of the in-network (push-down) aggregation approaches described above for a simple network topology. In the best case, every sensor only needs to send one merged data packet to the next hop in each round, no matter how many sensors are in the network. The packet merge curve increases slightly as intermediate packets get larger as the number of nodes grows. Without in-network aggregation, a node n has to send a data packet for each node whose route goes through n, so energy consumption increases very fast.

We also investigated the effect of in-network aggregation on the delay of receiving the answer at the gateway node, as shown in the middle graph in Figure 38.7. When the network size is very small, in-network aggregation introduces little extra delay due to synchronization; however, as the network size increases, direct delivery induces much larger delay due to frequent conflicts of packets at the MAC layer. The rightmost graph in Figure 38.7 shows the cumulative distribution of the percentage of received sensor readings versus the delay. Without push-down aggregation, we continously receive data, whereas with push-down aggregation we receive the final result in one packet.

© 2005 by Chapman & Hall/CRC

38.5 Related Work

Related work on query processing for sensor networks can be classified into two broad areas: work on *ad hoc* networks and distributed query processing.

38.5.1 *Ad Hoc* Networks

Research on routing in *ad hoc* wireless networks has a long history [27,28], and a plethora of papers has been published on routing protocols for *ad hoc* mobile wireless networks [29–35]. All these routing protocols are general routing protocols and do not take specific application workloads into account. The SCADDS project at USC and ISI explores scalable coordination architectures for sensor networks [36], and their data-centric routing algorithm, called directed diffusion [25], first introduced the notion of filters that we advocate in Section 38.3.3.

Ramanathan and Rosales-Hain [37] developed protocols for adjusting transmit power in *ad hoc* networks with the goal to improve connectivity in sparse networks and collisions in dense networks. PicoNet proposes an integrated design of radios, small, battery-powered nodes, and MAC and application protocols that minimize power consumption [38]. Nodes are also scheduled to turn their radios on and off in order to conserve energy; however, when a node needs to send a message to its neighbor it has to stay awake until it hears a broadcast message announcing its neighbor's reactivation. IEEE 802.11 supports *ad hoc* network configuration and provides power management controls [24]. Pearlman *et al.* [39] propose an energy-dependent participation scheme, where a node periodically re-evaluates its participation in the network based on the residual energy in its battery. GEAR [40] uses energy-aware neighbor selection to route a packet towards a target region and restricted flooding to disseminate the packet inside the destination region; it addresses the problem of energy conservation from a routing perspective without considering the interplay of routing and node scheduling.

An energy-efficient MAC protocol called S-MAC has been proposed by Ye *et al.* [41,42], where the nodes are locally synchronized to follow a periodic listen and sleep scheme. Each node broadcasts its schedule to its neighbors; the latter know exactly the time interval that the node is listening and (unlike PicoNet) only awake to send messages at that time. S-MAC does not explicitly avoid contention of the medium, but rather relies on an IEEE 802.11-style MAC for resolving collisions. This means that the receiving node must remain in the listening state for the (worst-case) time interval needed to resolve collisions, even its neighbors have no messages to send. Geographical Adaptive Fidelity (GAF) [43] is an algorithm that also conserves energy by identifying nodes that are equivalent from a routing perspective and then turning off unnecessary nodes.

38.5.2 Distributed Query Processing

An energy-efficient aggregation tree using data-centric reinforcement strategies is proposed Intanagonwiwat and co-workers [25,44]. In a recent study [45], an approximation algorithm has been designed for finding an aggregation tree that simultaneously applies to a large class of aggregation functions. A two-tier approach for data dissemination to multiple mobile sinks is discussed Ye *et al.* [46].

There has been a lot of work on query processing in distributed database systems [47–51], but as discussed in Section 38.1, there are major differences between sensor networks and traditional distributed database systems. Most related is work on distributed aggregation, but existing approaches do not consider the physical limitations of sensor networks [52,53]. Aggregate operators are classified by their properties by Gray *et al.* [20], and an extended classification with properties relevant to sensor network aggregation has been proposed by Madden *et al.* [19]. Other relevant areas include work on sequence query processing [54,55], and temporal and spatial databases [56].

Owing to space constraints, in this chapter we only introduced work on query processing, but there has been complementary recent work on in-network storage management. Ratnasamy and co-workers [57,58] propose a distributed storage model based on a hashing scheme; this model is extended by

© 2005 by Chapman & Hall/CRC

Ghose *et al.* [59]. Long-term storage in sensor networks is combined with multi-resolution data access and spatiotemporal data mining by Ganesan *et al.* [60].

38.6 Concluding Remarks and Future Challenges

We believe that a database approach to sensor network data management is very promising, and our initial experience with users of our technology has corroborated this belief: declarative queries offer the dual benefits of an easy-to-interface and an energy-efficient execution substrate. Furthermore, our approach has unearthed a plethora of interesting additional research problems in this domain. Thus, we conclude this chapter by listing some future challenges that we believe are important unsolved problems in sensor network query processing.

38.6.1 Adaptivity

In the context of query processing, adaptivity usually refers to the modification or rebuilding of query plans based on run-time observations about the performance of the system. Adaptivity will be required to maximize the longevity and utility of sensor networks; examples of adaptation opportunities include:

- *Query reoptimization*. Traditional query optimization, where queries are optimized before they are executed, is not likely to be a good strategy for long-running continuous queries, as noted by Madden *et al.* [5]. This is because statistics used to order operators may change significantly over the life of a query, as we discuss below.
- *Operator migration*. Operators in a query plan may be placed at every node in the network, or only at a few select nodes. Choosing which nodes should run particular operators is tricky; for example, a computationally expensive user-defined selection function might significantly reduce the quantity of data output from a node. Conventional wisdom suggests that this filter should be propagated to every node in the network. However, if such a filter is expensive enough, then running it may consume more energy than would be saved from the reduction in communication, so it may be preferable to run the filter at only a few select nodes (or at the powered base station). Making the proper choice between these alternatives depends on factors such as the local selectivity of the filter, the number of hops which nonfiltered data must travel before being filtered, processor speed, and remaining energy capacity.
- *Topology adjustment*. Finally, the system needs to adapt the network topology. Existing sensor-network systems generally perform this adaptation based on low-level observations of network characteristics, such as the quality of links to various neighboring nodes [61], but a declarative query processor can use query semantics to shape the network. For example, topology adaptation would be beneficial when computing grouped aggregates, since the maximum benefit from in-network aggregation will be obtained when nodes in the same group are also in the same routing sub-tree.

38.6.2 Nested Queries, Many-to-Many Communication, and Other Distributed Programming Primitives

TinyDB and Cougar support a limited form of nested queries through `STORAGE POINTS` — queries can be executed over the logged data in these buffers, thus providing a simple form of nesting. However, more complex, *correlated queries* that express fairly natural operations like ''find all the sensors whose temperature is more than two standard deviations from the average'' cannot be executed efficiently within the network. The reason for this is that this is really a nested query consisting of an inner query that computes the average temperature and an outer query that compares every node's temperature with this average. To execute this query in-network, some mechanism for propagating the average to the sensors is needed.

© 2005 by Chapman & Hall/CRC

One important question is the location of storage points. Where should we place storage points in order to balance resource usage across the network while minimizing overall network resource usage?

The natural solution is a two-round, gather–scatter protocol where some leader node (probably the root of the network) collects (gathers) the average and standard deviation and then disseminates (scatters) them to the entire network to answer the query. In-network processing can be a saving in this case because few sensors will have readings that are more than two standard deviations away from the average, and both average and standard deviation are algebraic aggregates that can be collected in an energy-efficient manner.

For other queries, still more sophisticated communication primitives are needed; for example, consider the query ''report all pairs of sensors within 1 m of each other that have the same vibration reading.'' Answering this query without bringing all of the sensor data out of the network requires sensors to be able to exchange information with their 1 m neighborhood, which should be relatively communication efficient but is a different communication pattern than the tree-based protocols we considered in this chapter.

Recently, in Cougar we proposed a simple technique for many-to-many communication in sensor networks based on the notion of a wave of communication flowing across a grid of sensors [62]. The main idea is to *schedule* sensors such that all collissions at the MAC layer will be avoided. Whitehouse and Sharp [63] propose a ''neighborhood abstraction'' that allows nodes to share information with their local neighbors. Incorporating these techniques into the implementations of TinyDB and Cougar and using them to support nested operations is an important area for future work.

38.6.3 Multi-Query Optimization

At any time, several long-running queries from multiple users might run over a sensor network. How can we share resources between these queries in order to balance and minimize overall resource usage? One possible approach is to apply standard multi-query optimization techniques from the literature, such as exploiting common subexpressions. Another intruiging approach is to allow user-specified *approximations* to queries, and thus enable approximations of queries through answers of other queries.

38.6.4 Heterogeneous Networks

So far we have only considered relatively homogeneous sensor networks, where all nodes are equally powerful. Future networks will likely have several tiers of nodes with different performance characteristics. How can SNQPs take advantage of this heterogeneity? For example, if we were to have a set of more powerful and stable sensor nodes, then they would seem to be excellent candidates for storage points.

Acknowledgments

Cougar has been partially funded by the Defense Advanced Research Project Agency under contract F30602-99-2-0528, NSF CAREER Grant 0133481, the Cornell Information Assurance Institute, and by a gift from Intel. TinyDB has been has been supported in part by the National Science Foundation under ITR/IIS grant 0086057, ITR/IIS grant 0208588, ITR/IIS grant 0205647, ITR/SI grant 0122599, and by ITR/IM grant 1187-26172, as well as research funds from IBM, Microsoft, Intel, and the UC MICRO program.

References

[1] Hill, J. *et al.*, System architecture directions for networked sensors, in *ASPLOS*, November 2000.
[2] Calimlim, M. *et al.*, Cougar project Web page, http://cougar.cs.cornell.edu.
[3] Madden, S. *et al.*, TinyDB Web page, http://telegraph.cs.berkeley.edu/tinydb.

© 2005 by Chapman & Hall/CRC

[4] Sakurai, T., Interconnection from design perspective, in *Proceedings of the Advanced Metallization Conference 2000 (AMC 2000)*, 2001, 53.
[5] Madden, S. *et al.*, Continously adaptive continuous queries over data streams, in *ACM SIGMOD*, Madison, WI, June 2002.
[6] Madden, S. and Franklin, M.J., Fjording the stream: an architechture for queries over streaming sensor data, in *ICDE*, 2002.
[7] Carney, D. *et al.*, Monitoring streams — a new class of data management applications, in *VLDB*, 2002.
[8] Motwani, R. *et al.*, Query processing, approximation and resource management in a data stream management system, in *CIDR*, 2003.
[9] Gehrke, J. *et al.*, On computing correlated aggregates over continual data streams, in *Proceedings of the ACM SIGMOD Conference on Management of Data*, Santa Barbara, CA, May 2001.
[10] Madden, S. *et al.*, The design of an acquisitional query processor for sensor networks, in *ACM SIGMOD*, 2003, in press.
[11] Chu, M. *et al.*, Scalable information-driven sensor querying and routing for *ad hoc* heterogeneous sensor networks, in *International Journal of High Performance Computing Applications*, 16(3), 293, 2002.
[12] Texas Advanced Optoelectronic Solutions, TSL2550 ambient light sensor, Technical report, September 2002, http://www.taosinc.com/pdf/tsl2550-E39.pdf.
[13] Intersema, MS5534A barometer module, Technical report, October 2002, http://www.intersema.com/pro/module/file/da5534.pdf.
[14] Sensirion, SHT11/15 relative humidity sensor, Technical report, June 2002, http://www.sensirion.com/en/pdf/Datasheet_SHT1x_SHT7x_0206.pdf.
[15] Melexis Microelectronic Integrated Systems, MLX90601 infrared thermopile module, Technical report, August 2002, http://www.melexis.com/prodfiles/mlx90601.pdf.
[16] Analog Devices, Inc. *ADXL202E: Low-Cost 2 g Dual-Axis Accelerometer*. http://products.analog.com/products/info.asp?product=ADXL202.
[17] Atmel Corporation. Atmel ATMega 128 Microcontroller Datasheet. http://www.atmel.com/atmel/acrobat/doc2467.pdf.
[18] Yao, Y. and Gehrke, J., Query processing in sensor networks, in *Proceedings of the First Biennial Conference on Innovative Data Systems Research (CIDR 2003)*, Asilomar, CA, January 2003.
[19] Madden, S. *et al.*, TAG: a Tiny AGgregation Service for *ad-hoc* sensor networks, in *OSDI*, 2002.
[20] Gray, J. *et al.*, Data cube: a relational aggregation operator generalizing group-by, cross-tab, and sub-total, in *ICDE*, February 1996.
[21] Honeywell, Inc., *Magnetic Sensor Specs HMC1002*, http://www.ssec.honeywell.com/magnetic/spec_sheets/specs_1002.html.
[22] Dawson, T., Fog in the California redwood forest: ecosystem inputs and use by plants. *Oecologia*, (117), 476, 1998.
[23] Breslau, L. *et al.*, Advances in network simulation. *IEEE Computer*, 33(5), 59, 2000.
[24] I.C. Society, Wireless LAN medium access control (mac) and physical layer specification, IEEE Std 802.11, 1999.
[25] Intanagonwiwat, C. *et al.*, Directed diffusion: a scalable and robust communication paradigm for sensor networks, in *Proceeedings of MOBICOM 2000*, ACM SIGMOBILE, ACM Press, 2000, 56.
[26] Gray, J. *et al.*, Data cube: a relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Mining and Knowledge Discovery*, 1(1), 29, 1997.
[27] Jubin, J. and Tornow, J.D., The DARPA packet radio network protocol, *Proceedings of the IEEE*, 75(1), 21, 1987.
[28] Schacham, N. and Westcott, J., Future directions in packet radio architectures and protocols, *Proceedings of the IEEE*, 75(1), 83, 1987.

© 2005 by Chapman & Hall/CRC

[29] Perkins, C. and Bhagwat, P., Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers, in *ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications*, August 1994, 234.

[30] Johnson, D.B. and Maltz, D.A., Dynamic source routing in *ad hoc* wireless networks, in Imielinski, T. and Korth, H. (eds), *Mobile Computing, The Kluwer International Series in Engineering and Computer Science*, volume 353, Kluwer Academic Publishers, 1996.

[31] Broch, J. *et al.*, A performance comparison of multi-hop wireless *ad hoc* network routing protocols, in *Proceedings of the 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM-98)*, ACM SIGMOBILE, ACM Press, 1998, 85.

[32] Perkins, C.E. and Royer, E.M., *Ad-hoc* on-demand distance vector routing, in *Workshop on Mobile Computing and Systems Applications*, 1999.

[33] Park, V. and Corson, S., Temporally-ordered routing algorithm (tora) version 1 functional specication, Internet draft, http://www.ietf.org/internet-drafts/draft-ietf-manet-tora-spec-02.txt, 1999.

[34] Das, S. *et al.*, Performance comparison of two on-demand routing protocols for *ad hoc* networks. in *Proceedings of the 2000 IEEE Computer and Communications Societies Conference on Computer Communications (INFOCOM-00)*, Los Alamitos, March 26–30, IEEE, 2000, 30.

[35] Johansson, P. *et al.*, Scenario-based performance analysis of routing protocols for mobile *ad-hoc* networks, in *Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom-99)*, ACM SIGMOBILE, ACM Press, 1999, 195.

[36] Estrin, D. *et al.*, Next century challenges: scalable coordination in sensor networks, in *Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom-99*), ACM SIGMOBILE, ACM Press, 1999, 263.

[37] Ramanathan, R. and Rosales-Hain, R., Topology control of multihop wireless networks using transmit power adjustment, in *Proceedings of the IEEE Infocom*, March 2000, 404.

[38] Bennett, F. *et al.*, Piconet: embedded mobile networking. *IEEE Personal Communications*, 4(5), 8, 1997.

[39] Pearlman, M.R. *et al.*, Elective participation in *ad hoc* networks based on energy consumption, in *Proceedings of the IEEE GLOBECOM*, 2002, 17.

[40] Yu, Y. *et al.*, Geographical and energy aware routing: a recursive data dissemination protocol for wireless sensor networks, Technical report UCLA/CSD-TR-01-0023, University of Southern California, May 2001.

[41] Ye, W. *et al.*, An energy-efficient MAC protocol for wireless sensor networks, in *Proceedings of the IEEE Infocom*, 2002, 1567.

[42] Ye, W. *et al.*, Medium access control with coordinated, adaptive sleeping for wireless sensor networks, Technical report ISI-TR-567, USC/Information Sciences Institute, January 2003.

[43] Xu, Y. *et al.*, Topology control protocols to conserve energy inwireless *ad hoc* networks, Technical report 6, University of California, Los Angeles, Center for Embedded Networked Computing, January 2003.

[44] Heidemann, J. *et al.*, Building efficient wireless sensor networks with low-level naming, in *SOSP*, October 2001.

[45] Goel, A. and Estrin, D., Simultaneous optimization for concave costs: single sink aggregation or single source buy-at-bulk, in *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, January 2003.

[46] Ye, F. *et al.*, A two-tier data dissemination model for large-scale wireless sensor networks, in *Proceedings of the Eighth Annual International Conference on Mobile Computing and Networking (MobiCom)*, 2002.

[47] Yu, C.T. and Chang, C.C., Distributed query processing, *ACM Computing Surveys*, 16(4), 399, 1984.

[48] Ceri, S. and Pelagatti, G., *Distributed Database Design: Principles and Systems*, MacGraw-Hill, New York, NY, 1984.

© 2005 by Chapman & Hall/CRC

[49] Ozsy, M.T. and Valduriez, P., *Principles of Distributed Database Systems*, Prentice Hall, Englewood Cliffs, 1991.
[50] Yu, C. and Meng, W., *Principles of Database Query Processing for Advanced Applications*, Morgan Kaufmann, San Francisco, 1998.
[51] Kossmann, D., The state of the art in distributed query processing, *Computing Surveys*, 32, 2000.
[52] Shatdal, A. and Naughton, J.F., Adaptive parallel aggregation algorithms, in *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, Carey, M.J. and Schneider, D.A. (eds), San Jose, California, 22–25 May 1995, 104.
[53] Yan, W.P. and Larson, P.-Å., Eager aggregation and lazy aggregation, in *VLDB'95, Proceedings of 21th International Conference on Very Large Data Bases*, Dayal, U. *et al.* (eds), Zurich, Switzerland, 11–15 September, Morgan Kaufmann, 1995, 345.
[54] Seshadri, P. *et al.*, seq: a model for sequence databases, in *Proceedings of the Eleventh International Conference on Data Engineering, March 6–10, 1995, Taipei, Taiwan*, Yu, P.S. and Chen, A.L.P. (eds), IEEE Computer Society, 1995, 232.
[55] Seshadri, P. *et al.*, The design and implementation of a sequence database system, in *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3–6, 1996, Mumbai (Bombay), India*, Vijayaraman, T.M. *et al.* (eds), Morgan Kaufmann, 1996, 99.
[56] Zaniolo, C. *et al.* (eds), *Advanced Database Systems*, Morgan Kaufmann, San Francisco, 1997.
[57] Ratnasamy, S. *et al.*, Data-centric storage in sensornets, in *First Workshop on Hot Topics in Networks (HotNets-I) 2002*, 2002.
[58] Ratnasamy, S. *et al.*, Ght: a geographic hash table for data-centric storage, in *First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)*, 2002.
[59] Ghose, A. *et al.*, Resilient data-centric storage in wireless *ad-hoc* sensor networks, in *Proceedings of the 4th International Conference on Mobile Data Management MDM 2003*, 2003, 45.
[60] Ganesan, D. *et al.*, Dimensions: why do we need a new data handling architecture for sensor networks? in *Proceedings of the First Workshop on Hot Topics in Networks (HotNets-I)*, Princeton, New Jersey, 2002.
[61] Woo, A. and Culler, D., A transmission control scheme for media access in sensor networks, in *ACM Mobicom*, July 2001.
[62] Trigoni, N. *et al.*, Wavescheduling: energy-efficient data dissemination for sensor networks, in Submission, June 2003.
[63] Whitehouse, K. and Sharp, C., A neighborhood abstraction for exploiting locality in sensor networks, in Submission, April 2003.

© 2005 by Chapman & Hall/CRC

39

Autonomous Software Reconfiguration

R.R. Brooks

39.1 Problem Statement

Sensor nodes are cheap, small, resource-constrained devices. Effective applications require decisions to be made (and actions to be taken) cooperatively by large groups of nodes. There are many good reasons:

- Nodes are placed in the immediate vicinity of a chaotic environment. Many nodes will be destroyed in the course of a mission.
- When nodes rely on battery power, they will die over time. Creating long sleep cycles for nodes can extend the lifetimes of each individual node.
- System robustness is aided by removing central points of failure.

In addition to these factors, node missions may change over time. There is an inherent contradiction. These requirements indicate the need for a large, complicated software infrastructure. On the other hand, this infrastructure would need to be implanted on nodes lacking the storage and cycles to support it. One way to overcome this would be to assign specific tasks to nodes, effectively spreading the software storage and execution burden over many nodes. The danger is that this added complexity would introduce brittleness to the system by creating multiple single points of failure.

This chapter motivates our use of mobile code to reconfigure software autonomously on individual nodes. Each node's role can change over time. Should a node performing a critical role be destroyed, another can take over that role. Should the system's environment change, e.g. new classes of entities for tracking are identified, then the logic for identifying the entity can be painlessly added to the system. This comes at the cost of increased network communications. A hard constraint on embedded systems is thus replaced with a soft trade-off.

39.2 Resource Constraints

Applications of embedded systems, sensor networks, and miniature robots are severely limited by constraints such as storage, power, computational resources, bandwidth, available sensors, and physical strength. Many of these constraints can be alleviated by using mobile code and data.

© 2005 by Chapman & Hall/CRC

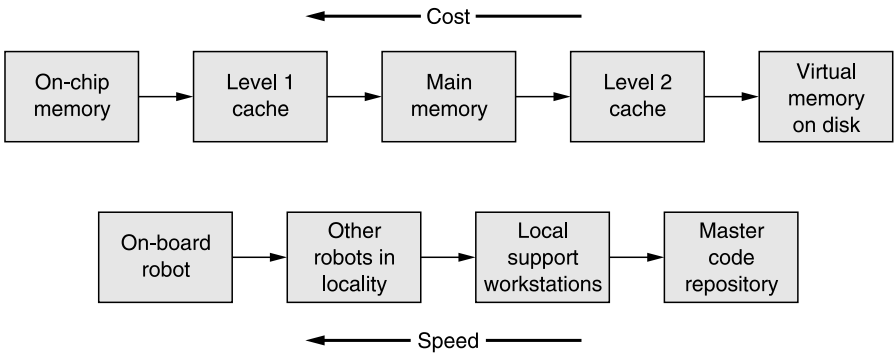


Figure 39.1. Memory hierarchies in virtual memory and in sensor networks.

Currently, a hard constraint is imposed by the number of behaviors that can be stored locally. The ability to upload behaviors provides a potentially unlimited range of behaviors constrained by the time needed to upload the behavior. This resembles a computer's use of virtual memory, where memory storage on disk alleviates physical memory limitations. The number of accessible behaviors will go from tens (currently) to thousands.

Storage is in a hierarchy of memories, each with varying costs and retrieval time. This provides the same benefits for robotic systems that virtual memory provides for computers. Figure 39.1 illustrates the different levels of memory in a traditional computer [1] and the proposed system. In both, memory access speed and cost increase at each level of the hierarchy. Computers are currently designed to allow access to as large an amount of information as possible, as quickly as possible, within cost constraints. When information is not available at any level in the hierarchy, an interrupt propagates the request for information to the next level, until the request is satisfied.

Mobile code can organize and autonomously reconfigure software services in the same way. A sensor network software service broker is an off-board repository of mobile code defining behaviors and supporting tasks. If a node N attempts to execute a behavior B that is not stored locally, a network interrupt is generated. The interrupt is propagated to the service broker and B is uploaded to N. B may be cached locally to reduce network traffic.

Access to software is no longer constrained by local memory (Figure 39.1). This allows behaviors to be available to autonomous systems for rare events. Systems can also be updated in response to changing circumstances. For example, target-recognition routines can be changed as information becomes available about the types of system the enemy has in the field.

Using mobile code also makes coordination simpler. Supervisory tasks are allocated to nodes at run time, based on current conditions such as proximity to the task. Failure of one node can be overcome by transferring responsibilities and behaviors to another one in the vicinity. Multiple sensing modalities can be accommodated at run time. We assume networked autonomous nodes with programmable microprocessors. They can download data and upload code. Mobile code and data support are layered onto existing miniature hardware.

The infrastructure for migrating code and data on demand has been implemented. It supports modelling arbitrary constraints, such as programs requiring specific processors, network connectivity, and hardware requirements. Figure 39.2 shows the roles in the current system.

39.3 Example Application Scenario

To support information dominance, unattended sensor network systems have been developed. New ones are being researched. They generally consist of ground-deployed, battery-powered, intelligent systems with sensors and wireless communications.

© 2005 by Chapman & Hall/CRC

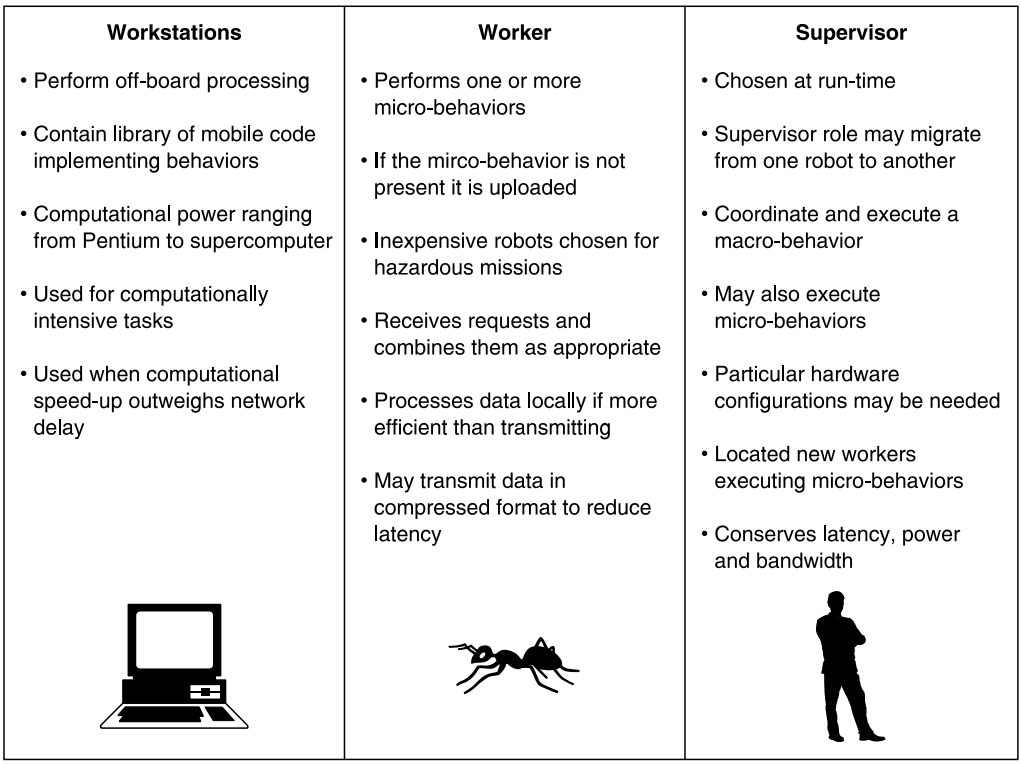


Figure 39.2. Workstations perform off-board processing. At run time some nodes may execute macro-behaviors supervising other nodes. Worker nodes execute micro-behavior components of higher level macro-behaviors. Multiple levels and specializations are possible.

Communications are difficult for objects on or near the ground. Multiple propagation paths cause signal interference, limiting the communications range. To compensate, systems use directional communications or create multi-hop networks. In multi-hop networks, each sensor node communicates with neighbors located in the immediate vicinity. A message travels from node to node, eventually reaching the information consumer.

For either approach, it is advantageous to place sensor nodes so that multiple communication paths are possible. The system is thus able to tolerate limited failures of single sensor nodes. This, and the use of multiple communications frequencies, can compensate somewhat for the presence of enemy jamming devices in the vicinity.

Since unattended ground sensors (UGSS) are often required in hazardous regions, it is reasonable to use unmanned vehicles to deploy them. Simultaneously deploying multiple sensor nodes requires coordination among multiple robots.

For this type of mission, it is reasonable to use networked autonomous vehicles with programmable microprocessors. They download data and upload code, as shown in Figure 39.3. Mobile code and data support are layered onto existing robots. It is necessary to model constraints, such as programs requiring specific processor types and data requests requiring specific sensor types.

An operator defines top-level behavior as a script of macro-behaviors. Behaviors are organized in a semiotic class structure reflecting the problem space. This structure is modeled by graphs of interacting automata [2]. Under-defined aspects of the script are resolved at lower levels of the hierarchy. The system is controlled recursively from the top down by macro-behaviors defining subsets of the network.

Macro-behaviors can be goals (e.g. *find_UGS*) or physics-based task definitions (e.g. *lift_UGS*(X)). A macro-behavior M_a is a script of macro-behaviors and micro-behaviors. M_a is supervised by part of

© 2005 by Chapman & Hall/CRC

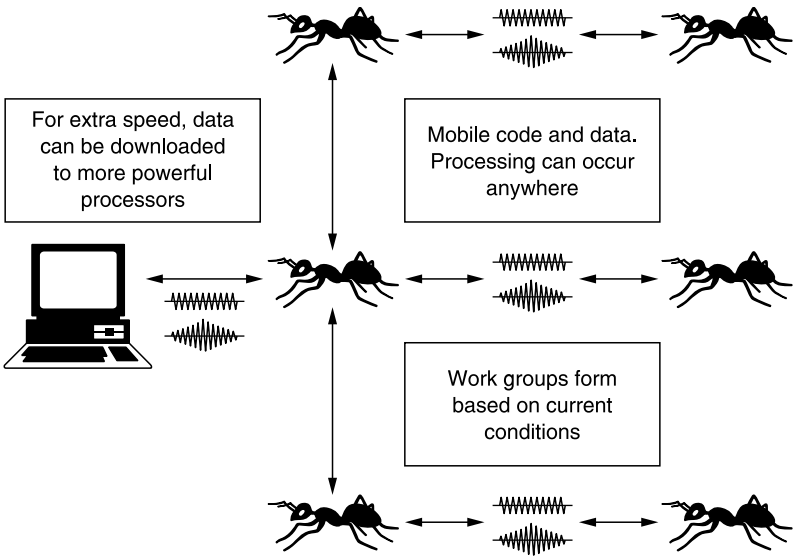


Figure 39.3. Mobile code allows the system to decide dynamically where processing should occur. Roles are assigned dynamically. Coordination and supervision are done locally. Sensor data travels up the network as needed. The entire network becomes an adaptive data-processing device.

the subset of robots executing M_a. A micro-behavior M_i is the mobile code for a specific behavior on a specific robot platform. For example, the class of behaviors *go_to*(X) contains many micro-behaviors. The micro-behavior needed by specific hardware is chosen at run time. Behaviors for robots mimicking crickets vary from behaviors for wheeled robots to swimming robots. The semiotic class structure resolves these ambiguities. An example of the utility of this approach is shown in Figure 39.3.

The system is also controlled from the bottom up by sensors providing information (*push*). Sensor interpretation is done by signal processing and sensor fusion [3] routines distributed throughout an active network.

To decide between alternative actions we use a *satisficing* approach, finding ''good enough'' alternatives in real-time by estimating performance as a function of resources, using experimental methods [4]. This has been endorsed by a number of researchers [4,5]. Factors to consider include time, memory, uncertainty, bandwidth, and result quality. Decisions consider *meta-reasoning* to estimate decision overhead. Special attention should be given to hardware cost. For hazardous missions, low-cost nodes can minimize loss.

39.4 Distributed Dynamic Linking

Assume we have a network with sensors attached to programmable microprocessors, and that those sensors can download data and upload code. With the libraries of mobile code that are available for use and the real-time networking control built into the system, this allows adaptation to changing conditions. The sensor network should also be generic to support multiple sensing modalities by conforming to standard frameworks for intelligent sensing systems [6,7]. The sensors integrate several phenomena, and physical phenomena are transmitted in a medium through an aperture to a detection device. The detected signals are then processed and decisions made [8].

Parts of our approach are appropriately implemented in a low-level operating system or network software. Communications are based on an asynchronous model with time-outs, but synchronous communication can be emulated.

A service broker is a system for organizing and invoking software services, and we have implemented one for mobile code support. We discuss how this infrastructure works for Java, Forth, and C++.

© 2005 by Chapman & Hall/CRC

Java's support of mobile code makes it relevant to sensor networks, but other aspects are poorly suited for embedded applications:

- The full Java virtual machine required 1.5 megabytes of storage in 1998 [9], which is excessive for some embedded applications. Since then, reduced virtual machines with partial functionality have been implemented.
- Embedded systems often require direct software interaction with hardware. Java's design intentionally omits this.
- Java's networking assumptions are not always appropriate.

Forth is commonly used for embedded programming. Like Java, it is interpreted. It is not object oriented. Forth was designed for modular task construction and interaction with hardware. Many interpreters for Forth have small memory profiles [10]. It has many of the favorable aspects of Java, without the unfavorable ones.With regard to C++, it is a widely available, compiled, object-oriented language with low-level constructs for hardware interaction. Java's syntax and structure are closely related to C++. C++ support provides completeness. Much of sensor-network-relevant code is written in C++ [3,11].

Java supports mobile code by compiling source code into a compressed byte code intermediate representation. Byte code is transmitted over the network to a virtual machine that executes it. In addition, a remote method invocation (RMI) protocol allows Java programs to execute routines on remote machines [12]. RMI is similar to the Object Management Group (OMG), Common Object Request Broker Architecture (CORBA), and Microsoft Distributed Common Object Model (DCOM).

Java's mobile code model is inappropriate for C++ and Forth. The service broker approach is language agnostic, layering mobile code support onto existing languages. This approach can be easily adapted to other interpreted and compiled languages. Figure 39.4 shows our approach to mobile code support. It contains:

- *Java.* Most necessary support is in the language definition. We require routines for locating code on demand.
- *Forth.* A Forth interpreter must be present on the machine for Forth programs to execute. Forth code is stored in clear text. Mobile code support requires routines that:
  - Locate programs on demand.
  - Download routines as needed.
  - Compress code for transfer.
  - Decompress code for execution.
  - Execute Forth programs remotely.
  - Return results over network.
  - Reclaim resources after processing.
- *C++.* C++ is compiled before execution. Source code, object code, or binary executables can be transmitted. Object and binary files are incompatible between machine types and operating systems. Storage limitations restrict the ability of programs to run on some nodes. Code transfer between machines may involve multiple steps: (1) source code copied from machine *A* to machine *B*; (2) code built on *B* with cross-compiler for machine *M*; (3) binary executable copied from *B* to M; and (4) program executes on *M*. Mobile code support requires routines that:
  - Locate programs on demand.
  - Locate compilers (or cross-compilers) for classes of machines.
  - Download routines as needed.
  - Compress code for transfer.
  - Decompress code for execution.
  - Execute C++ programs remotely.

© 2005 by Chapman & Hall/CRC

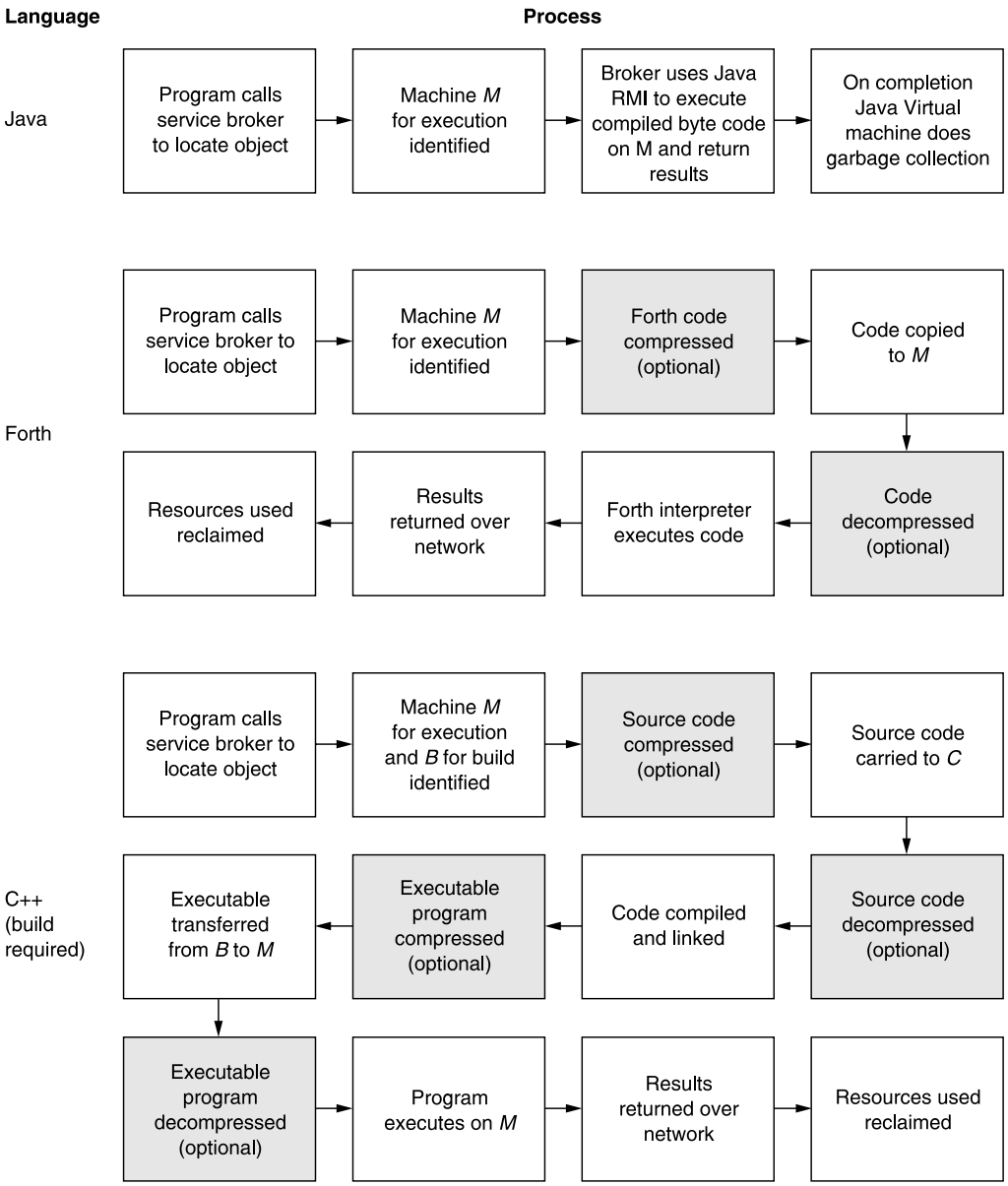


Figure 39.4. Typical processing performed by the service broker to invoke a service. More steps exist when calls are bound to service classes. Forth and Java portions are fairly standard. C++ libraries may consist of source, object, or executable files. C++ processing is shown for the case where the library stores source code.

- Return results over network.
- Reclaim resources after execution.

Libraries of mobile code can be developed and managed using this distributed service broker. When developing programs, designers bind calls to services or classes of services. If a call is bound to a class, then the service broker chooses a service in the class at run time to use resources efficiently given the current conditions. The service is chosen using profile functions derived using empirical methods. Resource-bounded optimization techniques make the choice. The amount of time spent choosing

© 2005 by Chapman & Hall/CRC

depends on the amount of time optimization can save. Time is unavailable for computing strictly optimal solutions on-line [13]. *Satisficing* solutions provide "good enough" answers [14], i.e. the best found given current resources and constraints.

For calls bound to classes, hardware and software configurations are chosen at run time. It is undesirable to force all services bound to the same class to use identical parameter lists. In addition, programs and data may or may not be located on the same node. For transparency and efficiency, parameter formats may have to be modified at run time. For these reasons, interfaces must be malleable.

Run-time support finds adequate services, determines the likelihood of services fulfilling process requirements, and assigns resources to services. Process overhead is part of the calculations. It is a waste of time for the broker to spend 5 min calculating how to save 3 min. Our approach uses meta-reasoning. Optimization has a limited horizon, due to the dynamic nature of the system. Groups of components form flexible *ad hoc* confederations to deliver data in response to changing needs and resources. This is applicable to any sensing modality and can be used on any system containing networked processors, sensors, and embedded processors.

Services are registered interactively. Service information needs are part of the information registered. Interface requirements, similar to CORBA interface definition language (IDL) interfaces [15], are mapped to existing class parameters, or a new class is defined. There need not be a one-to-one correspondence between class and service parameters.

At design time, nodes where programs and data are located are undefined. Communication must be transparent to distributed processing. For parameter passing by value, this is not an issue. Parameter passing by reference can be difficult to implement. When data and program are on the same node, pass by reference should be used for efficiency. When they are on different nodes, pass by reference can usually be emulated with the ADA read–write parameter mechanism. The called program executes and the results are written into the variable's storage. They are not always equivalent, and work is needed to insure consistency in all cases.

Binding links to classes of services causes another problem. It is simpler to design a service broker by insisting all services in a class have an identical interface. This is restrictive for service providers. When registering methods, links are made between service and class parameters. This mapping must not always be one-to-one. Defaults may or may not be provided; their use is encouraged and supported. Similarly, consumers make links between program objects and class parameters. At run time, consumer and provider interfaces negotiate exact parameter passing methods. A protocol exchanges interface information. Inconsistencies may be found on both sides and alternatives proposed. This includes issues like data format (e.g. integer variables may need to be converted to floating point). The process iterates until agreement is reached. Similar issues exist in inter-program communication for the program–program interface. The system emulates established methods using an asynchronous message-passing protocol.

The final issue we consider is adaptation to system state. Information can be exchanged in a number of formats. It is reasonable to compress data for transmission over slow channels. Transmission over noisy channels requires redundant data. As noise (lack of noise) is detected in a channel, error checking increases (decreases). The meta-protocol starts with pessimistic assumptions of channel quality and modifies the protocol dynamically. Modifications are based on information from normal operations. Extra traffic for monitoring status is avoided.

39.5 Classifier Swapping

As a proof-of-concept demonstration of the mobile code daemon, we recently tested a system for swapping classification and tracking algorithms on the fly. This allows the sensor network to adapt to changes in its environment. We implemented mobile code support for sensor networks by creating a minimal infrastructure.

Target classification is a process in which sensor time-series data are used to assign target detections to one of a set of known classes. A vehicle driving by a sensor node could be a tank, a truck, a dragon

© 2005 by Chapman & Hall/CRC

wagon, a TEL, etc. Many classification techniques exist. In this test we used classifiers developed by Hairong Qi of the University of Tennessee at Knoxville, Akhbar Sayeed of the University of Wisconsin [16], and David Friedlander of the Penn State Applied Research Laboratory [17].

To choose between classifiers, confusion matrices [18] were used. A confusion matrix is a matrix where the rows are the actual target classes and the columns are the predicted target classes. Each element of the matrix $e_{i,j}$ expresses the probability that the classifier returns codebook value j when target type i was actually present. For example, the matrix

$$\begin{bmatrix} 0.90 & 0.10 \\ 0.25 & 0.75 \end{bmatrix}$$

could express the uncertainty in a classifier with two classes: tracked vehicle (class 1) and wheeled vehicle (class 2). In this case, the system correctly classifies the class-1 vehicle 90% of the time and the class-2 vehicle 75% of the time. In our case, the codebook values and target classes used were:

Codebook	Target name	Target descriptor
0	Unknown	
10	sif_Buzzer	
11	sif_Motorcycle	Motorcycle
12	sif_TruckGas	Pickup truck, gas engine
13	sif_TruckDiesel	Pickup truck, diesel engine
14	sif_BuzzerRed	Red team
15	sif_BuzzerBlue	Blue team

Another part of the field test tested new classification techniques. Insufficient data were available before the tests to derive reliable confusion matrices for the three approaches used. Our classifier swapping tests used matrices fabricated to best illustrate the software functionality.

To support classifier swapping, each node keeps a vector containing the set of target classes most recently detected by the sensor node.[1] The diagonal of the confusion matrix expresses the likelihood that a classifier is correct. The vector of target classes seen recently is multiplied against the diagonal of the confusion matrices for the three participating classifiers. This provides a measure of how well a given classifier should work given the current target mix.

All three participants used a unified classifier application programming interface (API). This integrated the three different classifiers into the tracking process via a single call to the mobile-code daemon. After each target was processed, the system determined which classifier was likely to work best with the current target mix. The daemon pre-fetches that classifier to be sure that it is present on the sensor node and uses it to classify the next target detected. This illustrates the distributed dynamic linking concept. A single classification call can trigger any of a number of implementations. The system automatically chooses the most appropriate one at run time.

Implementation of this approach requires passing data to and from the classification program or library routine. The mobile-code daemon can replace the routine used at will. We circumvent functionality traditionally given to the linker. We have used two different approaches to this problem: manufacturing call stacks and marshalling data to disk.

For the Windows 2000 and Windows CE versions of the daemon, we experimentally determined the layout of call stacks used by compilers. Calls to the daemon contain the data passed to the mobile-code

[1]We implicitly assume that the types of target the node is likely to see in the near future resemble what it has seen lately (locality in time and space).

© 2005 by Chapman & Hall/CRC

package. The daemon manufactures the appropriate call stack and passes it directly to the library routine. Passing data to standalone executables is done using a command-line interface.

For the Linux port of the software, we took a different approach. A small wrapper process is integrated around calls to the daemon. Before calling the daemon, necessary data are marshaled (written to disk). The daemon executes the desired classification routine within another wrapper routine, which first reads the marshaled data into memory.

In the future, we plan on retaining the marshaling approach. It is more flexible and easier to integrate with multiple compilers. It also allows us to pass data transparently between nodes, as long as the program does not rely on the actual physical location in memory of the data.

As of this writing, the individual software components have all been written and tested. Some issues arose in porting parts of the system from the Intel X86 architecture to the SH4 architecture, which delayed the unified test. Plans exist to test the integrated system with field data in the immediate future.

Given a set of target tracks and a set of current detections, it is necessary to assign detections to tracks. This problem is known as *data association*. A number of methods have been developed for data association and none has been found to be clearly superior.

One widely accepted approach is multi-hypothesis data association [19]. For each track, multiple hypotheses are maintained regarding the target's codebook value and dynamics. Tracks are maintained using the most likely interpretation of a target's motion at any point in time.

The data association technique used is much simpler. For each detection, we extrapolate candidate tracks forward in time assuming there is no change in target heading or velocity. We compare detection position, heading, and velocity with the extrapolated data using a Euclidean metric. This essentially uses a nearest-neighbor approach. Each detection is mapped to the closest track.

Brooks *et al.* [16] show how target dynamics can be built into the track extrapolation process using an extended Kalman filter. Any highly distributed data association approach will not be able to enforce global consistency as well as the centralized approaches do, e.g. like Bar-Shalom and Li [19].

The nearest-neighbor approach is straightforward and works well in many cases. In spite of this, it would be useful to use different techniques for track maintenance of different target classes. Different classes may have radically different dynamics. We have attempted to do this by integrating multiple trackers into our framework. Each class can be handled by a different tracking implementation. And the tracker implementation is responsible for the consistency of its own track.

As a demonstration of this concept, we integrated our tracking approach with a mobile-agent tracking approach implemented by Hairong Qi at the University of Tennessee Knoxville. This application of distributed dynamic linking allows the sensor network to track a virtually unlimited number of target classes. This is possible in the same way that virtual memory systems provide modern workstations with a virtually unlimited amount of core memory. At any point in time, any node can recognize only a limited number of target classes. But the set of classes can be modified dynamically over the network. A hard constraint caused by local storage restrictions has been replaced by a soft constraint caused by network resource restrictions.

Note that, when targets are sparse, multiple target tracking becomes a set of disjoint single-target tracking problems. The ability of any system to track a dense cloud of targets will be limited by the sensor's abilities to differentiate between independent target detections.

39.6 Dependability

Our technology allows node roles to be chosen on the fly. This significantly increases system robustness by allowing the system to adapt to the failure of individual nodes. The nodes that remain exchange readings and find answers.

Consider a heading and velocity estimation approach using triangulation [20–22] at least three sensor readings are needed to get an answer. In the following, we assume all nodes have an equal probability of failure q. In a nonadaptive system, when the ''cluster head'' fails, the system fails.

© 2005 by Chapman & Hall/CRC

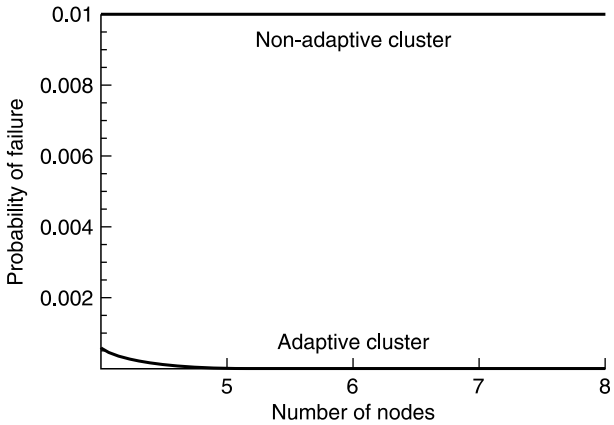


Figure 39.5. The top line shows the probability of failure for a nonadaptive cluster. The bottom line shows the probability of failure for an adaptive cluster. The probability of failure for a single node q is 0.01. The number of nodes in the cluster is varied from four to eight.

The cluster has a probability of failure q no matter how many nodes are in the cluster. In the adaptive case, the system fails only when the number of nodes functioning is three or less.

Figures 39.5–39.7 illustrate the difference in dependability between adaptive and nonadaptive tasking. These figures assume an exponential distribution of independent failure events, which is standard in dependability literature. The probability of failure is constant across time. We assume that all participating nodes have the same probability of failure. This does not account for errors due to loss of power.

In Figures 39.5 and 39.6 the top line is the probability of failure for a nonadaptive cluster. Since one node is the designated cluster head, when it fails the cluster fails. By definition, this probability of failure is constant. The lower line is the probability of failure of an adaptive cluster as a function of the number of nodes. This is the probability that less than three nodes will be available at any point in time. All individual nodes have the same failure probability, which is the value shown by the top line. The probability of failure of the adaptive cluster drops off exponentially with the number of nodes. Figure 39.7 shows this same probability of failure as a function of both the number of nodes and the individual node's probability of failure.

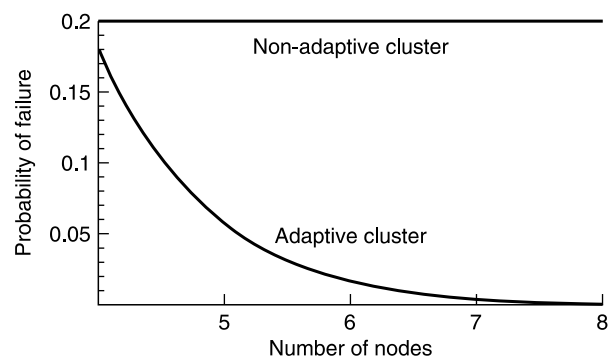


Figure 39.6. The top line shows the probability of failure for a nonadaptive cluster. The bottom line shows the probability of failure for an adaptive cluster. The probability of failure for a single node q is 0.2. The number of nodes in the cluster is varied from four to eight.

© 2005 by Chapman & Hall/CRC

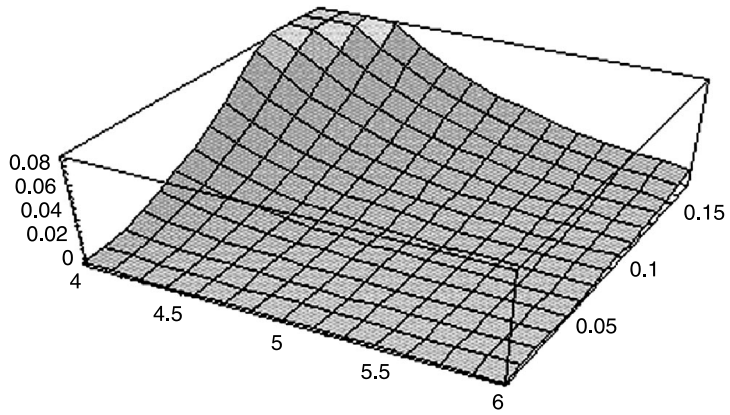


Figure 39.7. The surface shows the probability of failure (z axis) for an adaptive cluster as the probability of failure for a single node q varies from 0.01 to 0.2 (side axis) and the number of nodes in the cluster varies from four to six (front axis).

39.7 Related Approaches

In this section we compare the technology presented here with research in resource-limited optimization and robotics.

Figure 39.8 illustrates important differences between our approach and existing interoperability frameworks. CORBA and RPC allow interoperability among programs on diverse computing platforms, written in different languages. In CORBA, an IDL program is written to make the conversion. It is assumed the developer knows the service that will be used *a priori*. The developer must have the IDL. A broker is called by the application to locate the service at run time. Microsoft's DCOM has expanded this approach, placing a finite-state machine between the broker and the service. Our service broker uses semantic networks to aid in locating services for use both at design time and run time.

Four main technologies exist for distributed program interoperability. The Distributed Computing Environment Remote Procedure Call (RPC), the OMG CORBA, Microsoft's DCOM, and Sun's Java RMI. All provide the infrastructure for a program on one machine to invoke a procedure that executes on another machine. This includes methods for parameter passing over computer networks.

RPC is a client–server technology, which does not embrace the object model [23]. CORBA is built on RPC concepts work and provides a standard for object definition and interaction. CORBA objects can be transient or persistent. They can be written in any of a number of languages, and invoked

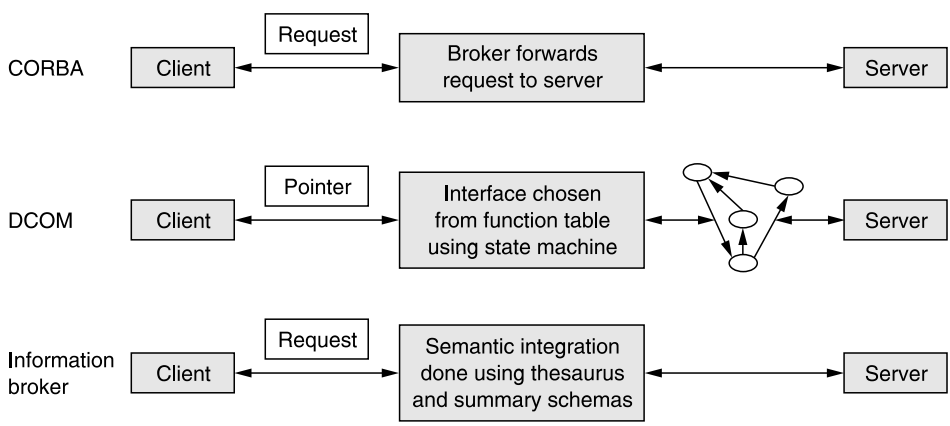


Figure 39.8. Contrast of the CORBA, DCOM, and info-broker integration models.

© 2005 by Chapman & Hall/CRC

transparently irrespective of their location on the network. Objects are called using their interfaces, which are written in an IDL. They register their existence with an Object Request Broker (ORB). To invoke a service, the calling routine needs the IDL definition. The calling routine sends a message to the ORB, which completes the connection [15].

DCOM is a binary-level standard defined by Microsoft. It differs from CORBA in that it specifies multiple interfaces to a single data object, using a pointer to a function table. Object state is made part of the interface. This approach is more scalable than a pure object-oriented inheritance hierarchy [24]. Details concerning DCOM implementation can be found in [25]. A detailed comparison of the use of CORBA and DCOM can be found in [26]. Interfaces exist for DCOM and CORBA interoperability.

Java is a distributed programming language developed by Sun Microsystems. Programs are compiled into a compressed byte-code. At run time the byte-code is downloaded and interpreted by a virtual machine. Part of the language is a standard for RMI. RMI is roughly equivalent to a CORBA ORB, in that programs can invoke procedures to execute on remote machines. As with DCOM, interfaces between Java and CORBA exist [27].

These interoperability implementations provide basic infrastructure for service brokering. OMG has also developed a standard for service brokering [12]. This standard allows a specific service to be chosen from a set of services of the same type at run time. The type name must be known *a priori*. Choice between services of the same type is made by Boolean comparisons of a set of parameters. This is a very limited method for comparison. These parameters may be dynamic, but dynamic parameter values must be obtained from the service at run-time. This can induce significant delays. Support is also provided for ORBs passing service requests to other ORBs. No semantic interoperability support is available for service discovery.

The FINDIT prototype attempts to combine multi-database systems with the world-wide-web. Databases have associated co-databases containing meta-information. Coalitions are formed by combining databases and meta-databases. The system is implemented on top of CORBA. Interfaces written in IDL are imported or exported. Only static meta-information, which is always true, is available about services. No mention is made of problems caused by use of differing terminology. Information is discovered by users browsing coalitions using the structured WebTassili query language [28,29].

Researchers at the TU Berlin have constructed a service broker for telecommunications applications that resembles the run-time portion of our proposal. Their work uses intelligent agent technology to implement an active network, allowing agents to work within telecommunications switches [30]. They divide agent technology research into the areas of mobile-agents and communication among static agents. They concentrate on mobile-agent technology. Another telecom broker implementation is discussed by Sharma *et al*. [31].

Xena is a service broker implemented at Carnegie Mellon University [32]. This also resembles the run-time portion of our proposal. This implementation concentrates on brokering for multiple services with different quality-of-service parameters. Services are chosen by user-defined functions. Little support is provided for discovering services, since ''general resource discovery is outside the scope of Xena.'' Global optimization is attempted using integer linear programming. Resource optimization is attempted by *semantic-preserving transformations*, such as converting MPEG to motion JPEG. For dynamic situations, autonomously performing local optimizations will provide better results, since global information will generally not be available.

Tanaka [33] describes a service broker prototype for telecommunications systems. Choices can be made among services of the same type. No support is provided for finding new types of service. Choices are made by comparing cost. More complicated choices are not supported. Yi *et al.* [34] describe a real-time service broker. Real-time support is provided through the use of time outs. No support is provided for semantic ambiguity.

Another service brokering system is TSIMMIS, which supports the retrieval of heterogeneous objects. Wrappers are called mediators. Translators are procedures that translate data formats [35]. Unfortunately, the assumption is made that attribute naming will be consistent across heterogeneous data sources. Research has shown this assumption to be questionable [36]. Semantic ambiguity is not

© 2005 by Chapman & Hall/CRC

compensated for. A structured query language similar to SQL is used for query construction. The retrieval model is more closely aligned to Web browsing than information indexing.

39.8 Summary

This chapter discussed autonomous software configuration for sensor networks. Motivation was given for the use of mobile code in networks of embedded systems. The concept of distributed dynamic linking was described in detail. Examples where this approach has been fielded were provided, as well as a dependability analysis of a sensor network with node roles chosen at run time. The analysis shows that system dependability can be greatly increased using this approach.

Acknowledgments and Disclaimer

This research is sponsored by the Defense Advance Research Projects Agency (DARPA), and administered by the Army Research Office under Emergent Surveillance Plexus MURI Award No. DAAD19-01-1-0504 and the Office of Naval Research under Award No. N00014-01-1-0859. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the sponsoring agencies

References

[1] Hwang, K., *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, McGraw-Hill, New York, NY, 1993.
[2] Phoha, S. *et al.*, A mobile distributed network of autonomous undersea vehicles, in *Proceedings of the 24th Annual Symposium and Exhibition of the Association for Unmanned Vehicle Systems International*, June, 1997.
[3] Brooks, R.R. and Iyengar, S.S., *Multi-Sensor Fusion: Fundamentals and Applications with Software*, Prentice Hall PTR, Upper Saddle River, NJ, 1998.
[4] Brooks, R.R. and Iyengar, S.S., Robot algorithm evaluation by simulating sensor faults, in *Proceedings of SPIE*, 2484, 394, 1995.
[5] Hooker, J.N., Needed an empirical science of algorithms, *Operations Research*, 42(2), 201 1994.
[6] Figueroa, F. and Mahajan, A., Generic model of an autonomous sensor, *Mechatronics*, 4(3), 295, 1994.
[7] IEEE Draft Standard for a Smart Transducer Interface for Sensors and Actuators, IEEE P1451.2, Draft 2.01, August 1996.
[8] Waltz E. and Llinas, J., *Multisensor Data Fusion*, Artech House, Boston, MA.
[9] McDowell, C.E. *et al.*, JAVACAM: trimming Java down to size, *IEEE Internet Computing*, May, 53, 1998.
[10] Tracy, M. *et al.*, *Mastering Forth*, Brady, New York, NY.
[11] Press, W.H. *et al.*, *Numeric Recipes in Fortran: The Art of Scientific Computing*, Cambridge University Press, Cambridge, UK.
[12] Object Management Group, Chapter 16 Trading Object Service Specification, CORBAServices, March 1997, http://www.omg.org/library/csindex.htm (last accessed on 7/23/2004).
[13] Zilberstein, S., Using anytime algorithms in intelligent systems, *AI Magazine*, Fall, 73, 1996.
[14] Bender, E.A., *Mathematical Methods in Artificial Intelligence*, IEEE Computer Society, Los Alamitos, CA, 1996.
[15] Seetharaman, K., The CORBA connection, *Communications of the ACM*, 41(10), 34, 1998.
[16] Brooks, R.R. *et al.*, Distributed target tracking and classification in sensor networks, in *Proceedings of the IEEE*, 91(8), 1163, 2003.

© 2005 by Chapman & Hall/CRC

[17] Friedlander, D. and Phoha, S., Semantic information fusion of coordinated signal processing in mobile sensor networks, *International Journal of High Performance Computing Applications*, 16(3), 235, 2002.
[18] Kohavi, R. and Provost, F., Glossary, *ML Journal*, 30(213), 272, 1998.
[19] Bar-Shalom, Y. and Li, X.-R., *Estimation and Tracking: Principles, Techniques, and Software*, Artech House, Boston, 1993.
[20] Brooks, R. *et al.*, Self-organized distributed sensor network entity tracking, *International Journal of High Performance Computer Applications*, 16(3), 207, 2002.
[21] Brooks, R. and Griffin, C., Traffic model evaluation of *ad hoc* target tracking algorithms, *International Journal of High Performance Computer Applications*, 16(3), 221, 2002.
[22] Moore, J. *et al.*, Tracking targets with self-organizing distributed ground sensors, in *2003 IEEE Aerospace Conference*, March 2003.
[23] Dogac, A. *et al.*, Distributed object computing platforms, *Communications of the ACM*, 41(9), 95, 1998.
[24] Wegner, P., Interoperability, *ACM Computing Surveys*, 28(1), 285, 1996.
[25] Gray, D.N. *et al.*, Modern languages and Microsoft's component object model, *Communications of the ACM*, 41(5), 55, 1998.
[26] Chung, P.E. *et al.*, DCOM and CORBA side by side, step by step, and layer by layer, http://www.bell-labs.com/~emerald/dcom_corba/Paper.html (last accessed on 7/23/2004).
[27] Curtis, D., Java, RMI and CORBA, http://www.omg.org/news/wpjava.html (last accessed on 7/23/2004).
[28] Benatallah, B. and Bouguettaya, A., Data Sharing on the Web, in *Proceedings of First International Enterprise Distributed Object Computing Workshop*, 1997, 258.
[29] Bouguettaya, A. *et al.*, Reflective data sharing in managing Internet databases, in *Proceedings 18th International Conference on Distributed Computing Systems*, 1998, 172.
[30] Breugst, M. and Magedanz, T., Mobile agents — enabling technology for active intelligent network implementation, *IEEE Network*, May/June, 53, 1998.
[31] Sharma, R. *et al.*, Environments for active networks, in *Proceedings of the IEEE 7th International Workshop on Network and Operating System Support for Digital Audio and Video*, 1997, 77.
[32] Chandra, P. *et al.*, in *Sixth International Workshop on Quality of Service (IWQoS 98)*, 1998, 187.
[33] Tanaka, H., Integrated environment for service-type repository management (IE-STREM), in *Proceedings of Global Convergence of Telecommunications and Distributed Object Computing TINA 97*, 1997, 363.
[34] Yi, S.-Y. *et al.*, Operating system support for the trader in distributed real-time environments, in *Proceedings of the Fifth IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems*, 1995, 194.
[35] Chawathe, S. *et al.*, The TSIMMIS project: integration of heterogeneous information sources, in *Proceedings of IPSJ Conference*, Tokyo, Japan, October 1994, 7.
[36] Heiler, S., Semantic interoperability, *ACM Computing Surveys*, 27(2), 271, 1995.

© 2005 by Chapman & Hall/CRC

40

Mobile Code Support

R.R. Brooks and T. Keiser

40.1 Problem Statement

Chapter 39 provided background on why autonomous software reconfiguration is needed for sensor networks. It also discussed applications we have fielded, and showed that this increases the system dependability. This chapter discusses our mobile-code daemon implementations in detail. Along with design details useful for implementing mobile code systems, we provide background on mobile code in distributed systems. This information helps provide a perspective on mobile code and how it can be integrated into distributed systems and ubiquitous computing implementations.

40.2 Mobile-Code Models

Von Neumann, a father of computer science, invented flow charts, programming languages, and the serial "von Neumann" computer [1]. His seminal concept of an automaton controlling another automaton [1] can be viewed as a form of mobile code. The mobile-code concept is evident in the 1960s remote job entry terminals that transfer programs to mainframe computers. Sapaty's wave system, created in the Ukraine [2] in the 1970s, provided full mobile-code functionality. In the 1980s, packet radio enthusiasts in Scandinavia developed a Forth-based approach to transfer and execute programs remotely through a wireless infrastructure [3]. In the 1990s, Java was developed by Sun Microsystems and became the first widely used mobile code implementation. Along the way, mobile code has been viewed from many different perspectives and paradigms.

Table 40.1 shows established mobile-code paradigms [4–7]. There has been a clear progression of technology. Initially the client/server approach supported running procedures on a remote node. Remote evaluation augmented this by allowing the remote node to download code before execution. Code on demand, as implemented in Java, allows local clients to download programs when necessary. Process migration allows processes to be evenly distributed among workstations. Mobile agents permit software to move from node to node, following the agent's internal logic. Active networks allow the network infrastructure to be reprogrammed by packets flowing through the network. Packets of data can be processed while being routed, through execution of encapsulating programs.

Paradigms differ primarily on where code executes and who determines when mobility occurs. Let us consider an example scenario. Given an input data file f on node n_f, a program p on node n_p to be

© 2005 by Chapman & Hall/CRC

Table 40.1. Common mobile-code paradigms

Paradigm	Example	Description
Client/server	CORBA	Client invokes code resident on another node
Remote evaluation	CORBA factory	Client invokes a program on remote node. Remote node downloads code
Code on demand	Java, Active X	Client downloads code and executes it locally
Process migration	Mosix, Sprite	Operating system transfers processes from one node to another for load balancing
Mobile agents	Agent-TCL	Client launches a program that moves from site to site
Active networks	Capsules	Packets moving through the network reprogram network infrastructure

executed, and a user u using node n_u, the paradigms in Table 40.1 would determine the following courses of action [5] (this problem is appropriate for neither process migration nor active networks):

- *Client/server*. Data file f is transferred from n_f to n_p. Program p executes on n_p and results are transferred to n_u.
- *Remote evaluation*. Program p is transferred to n_f and executes there. Results are returned to n_u.
- *Code on demand*. Data file f and program p are transferred to n_u and execute there.
- *Mobile agents*. Program p is transferred to n_f and executes there. Program p carries the results to n_u.

Each approach will be efficient at times, depending on network configuration and the size of p and f. The model we propose accepts all of these alternatives. In this model, nodes, files, etc. can be specified at run time.

An important distinction exists between *strong* and *weak* code mobility [5]. *Strong* mobility allows migration of both code and execution state. Programs can move from node to node while executing. This migration may even be transparent to the program itself (i.e. the program is not aware that it has migrated). *Weak* mobility transfers limited initialization data, but no state information, with the code.

The utility of strong migration is debatable, since it increases the volume of data transmitted as a process migrates [8]. For load balancing, strong migration is worthwhile only for processes with long lifetimes [9]. Mobile agents can be implemented using either weak or strong mobility. Differences of opinion exist in the literature as to whether distributed systems that handle migration transparently are [6] or are not [5] mobile-code systems. We consider them to be mobile-code systems.

In spite of the differences listed thus far, all mobile-code systems have many common aspects. A network-aware execution environment must be available. For Java applets, a Web browser with a virtual machine downloads and executes the code [10]. A network-operating system layer coupled with a computational environment provides this service for other implementations [5].

Some specific mobile-code implementations are difficult to fit into the paradigms in Table 40.1 and warrant further discussion:

- *Postscript*. This is one of the most successful mobile-code applications, but it is rarely recognized as mobile code. A postscript file is a program that is uploaded to a printer for execution. It produces graphic images as results.
- *Wave*. This may be the earliest successful implementation of network-aware mobile code. It was implemented in the Ukraine in the early 1970s [2]. Wave is a programming environment based on graph theory. Network nodes correspond to graph nodes. Network connections correspond to edges. Since distributed computing problems are often phrased in graph-theoretical terms [11], it is a very elegant approach.
- *Tube*. This extends a LISP interpreter to distributed applications [12]. As an interpreted system, LISP is capable of meta-programming. Code can be generated and modified on the fly. The distributed interpreter is capable of robust computations and compensating for network errors.

© 2005 by Chapman & Hall/CRC

- *Messenger*. This uses mobile code as the backbone for implementing computer communications protocols [13]. Protocol data units (PDUs) are passed from transmitters to receivers, along with code defining the meaning of the PDUs. This concept is similar to active networks [7], but the approach is very different. Instead of concentrating on the mechanics of communication, this approach looks at the semiotics of message passing.
- *Jini*. This adds distributed services, especially a name service, to the Java remote method invocation (RMI) module [14]. Objects can be invoked on remote nodes. Jini is intended to extend Java technology to smart spaces, ubiquitous computing, and embedded systems. NIST's smart-space researchers have found difficulties with this approach, since the Java virtual machine's 1.5-megabyte memory footprint [15] is larger than the address space of most widely used embedded processors [16].

The paradigms discussed have been primarily oriented towards producing prototypes or commercial applications, rather than establishing the consequences of code mobility.

40.3 Distributed Dynamic Linking

In our work [17] we have developed methods to support transparent use of equivalent services with different implementations on heterogeneous machines. These concepts, *polymorphism* and *distributed dynamic linking*, are described in this section. We allow daemons to choose nodes for executing tasks as a function of the current network state. They are also enabled to choose between software implementations.

For code to migrate to a remote node, an active process must listen to the network on the remote node. This can be a user (Java applet invocation), or system (Java RMI) process. We use lightweight network daemons for this purpose [17]. The system currently runs on Windows NT, Windows CE and Linux. Daemons accept remote requests for code transmission and execution. Data are transferred as needed. Daemons also monitor the local system state (work load, disk space, congestion, etc.). Remote code can be identified explicitly by a URL, or implicitly. In the current implementation, mobile code is cached locally. When a call is made to execute a mobile-code service, the daemon automatically retrieves it if it is not present. We have created this process, known as *distributed dynamic linking* [18]. The following calls allow software to manipulate the local software configuration directly:

- Verify local presence of mobile code modules.
- Pre-fetch mobile code for future use.
- Lock mobile code into local working set.

Data are transparently transferred between nodes by the daemon. Blocking and nonblocking calls are supported. A pipe abstraction in the API allows creation of distributed data-flow scripts on the fly. A program p_1 runs on node n_1 and its output o_1 is sent over the network to node n_2, triggering execution of program p_2 on n_2 using o_1 as input.

Code is grouped in classes. Each class contains mobile-code module implementations of a service [18]. Different implementations of the same service may exist in the repository. They may be executables or object libraries created for different operational environments, or they may be different algorithms for the same problem. When a call is made to execute a specific service on a specific node, the repository determines which implementation is appropriate for the target environment. This is equivalent to *polymorphism* in object-oriented languages, where different objects may be required to perform equivalent tasks. We have extended the concept to distributed environments and use it to tolerate heterogeneous hardware and operating systems. This project will extend the distributed dynamic linking and polymorphism concepts to create a single operational environment linking embedded processors, workstations, and supercomputers. The environment can choose appropriate software, hardware, and connecting for accomplishing a task.

Invocation information and a profile are stored with each mobile-code module. The profile expresses module capabilities as a function of resources used and as relevant constraints. The current

© 2005 by Chapman & Hall/CRC

implementation requires a target machine to be named by the calling program for execution of a mobile-code module. It should be possible for daemons to choose the machine at run time. Similarly, empirical methods characterize the resource needs of a given implementation in the profile.

Profiles model the relationship between time, system resources, and answer quality. Sun's law for speed-up:

$$S_n^* = \frac{W_1 + G(n)W_n}{W_1 + G(n)W_n/n}$$

(S_n^* is the speed-up achieved with n processing elements (PEs), W_j is the workload with degree of parallelism j, n is the number of PEs, and $G(n)$ is workload increase as memory increases) gives a theoretical basis for this relationship [19]. Sun's law shows that program speed-up is a function of the number of PEs, available memory, and problem size. Choices between alternative implementations are made by finding the implementation whose profile uses the least resources, for the best results, given the current conditions.

These methods allow us to start a task on any machine and have the machine construct a virtual enterprise to finish the task. The virtual enterprise recruits underutilized machines on the network. We create a *satisficing* response to the problem of how to allocate resources to the task [20]. The response is not optimal, but feasible and reasonable.

40.4 Daemon Implementation

The mobile-code daemon we present is based upon a core network protocol called the remote execution and action protocol (REAP). This protocol is responsible for message passing between nodes within our network. On top of this packet protocol we have developed a framework to allow objects to serialize themselves and travel across the network. At a higher layer of abstraction we have written messages to handle remote-process creation and monitoring, simple filesystem operations, and resource index operations. Use of the mobile-code daemon in sensor network applications is documented by Brooks and co workers [17,21].

The daemon is written in C++. The first version ran on the Windows NT and Windows CE operating systems. It has since been ported to the Linux operating system. The daemon structure (Figure 40.1) is broken down into several core modules: foundation classes, the networking core, the random graph module, the messaging core, the packet router, the index server, the transaction manager, the resource manager, and the process manager. We will discuss each of these components in turn.

Before discussing the REAP daemon in detail, it is useful to discuss the underlying framework on which it is built. The framework abstracts many of the complexities of systems programming out of the core into a set of libraries. Thus, we have written our own object-oriented threading and locking classes, whose current implementation calls into the threads library of the underlying operating system. We also rely heavily on a set of templated, multithreaded linked list, hash, and heap objects throughout the code. In addition, there are classes to handle singleton objects, the union-find problem, and object serialization. Lastly, there is also a polymorphic socket library that allows different networking architectures to emulate unicast stream sockets, regardless of the underlying network protocol or topology. These socket libraries are explained in the discussion of the networking core.

The daemon is capable of communicating over several networking technologies. The major ones are: Transmission control protocol/Internet protocol (TCP/IP), diffusion routing [22], and UNIX domain sockets. The socket framework is designed so that new protocols are easily inserted into the daemon. To achieve this, an abstract base class \verb'Socket' includes all of the familiar calls to handle network I/O. Furthermore, all nodes are assigned a protocol-independent unique address. Opening a new socket involves looking up the network-layer address of a node in a local cache, and then opening the lower level socket. When a cache miss occurs, a higher level protocol is provided to find the network-layer

© 2005 by Chapman & Hall/CRC

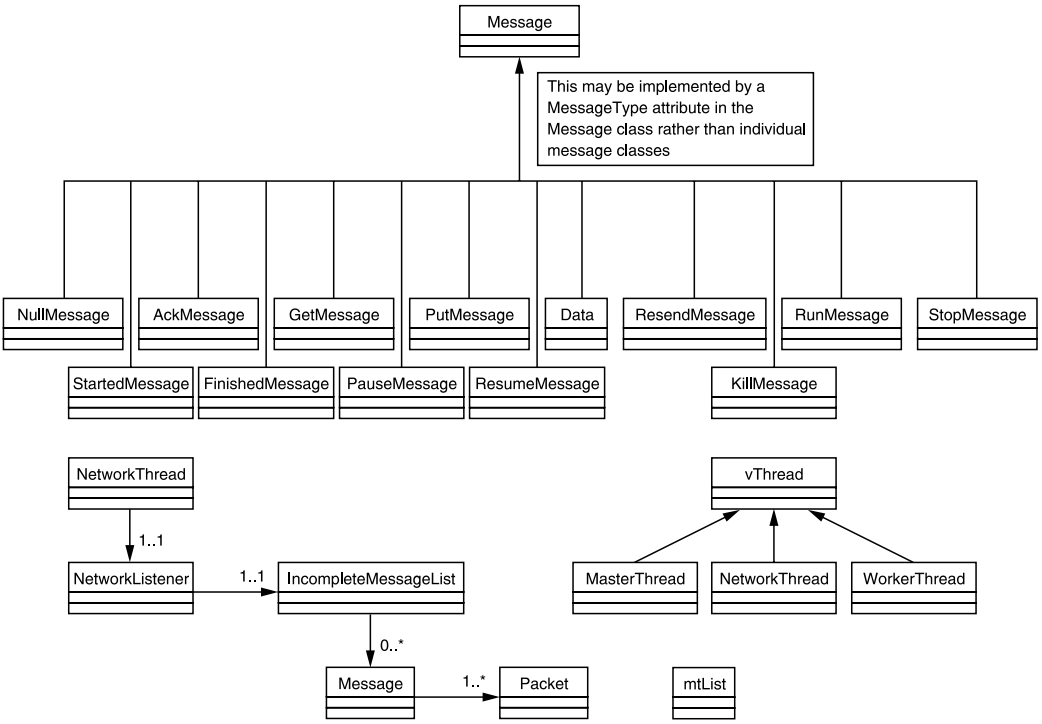


Figure 40.1. UML class structure of our current mobile-code daemon.

address. The appropriate socket object is allocated based upon the network-layer address of the destination.

Diffusion provided some interesting challenges because it is not a stream-oriented unicast protocol. Rather, it provides a publish and subscribe interface, and is essentially a multicast datagram protocol. Thus, we had the choice of rewriting the REAP socket protocol as a datagram protocol, or building a reliable stream protocol on top of the diffusion framework. It was deemed simpler to write a reliable stream protocol on top of diffusion. In essence, we wrote a simplified userspace TCP stack. The current userspace stack employs the standard three-way handshake protocols for socket open and close, and it also employs a simple delayed-ACK algorithm. This system is implemented as an abstract child of the 'Socket' base class. Our diffusion driver then provides an implementation of our userspace TCP module. The diffusion driver performs a role equivalent to the IP layer processing code in most kernels. It receives datagrams from the diffusion daemon through callback functions, parses the headers to make sure the datagram has reached the correct destination, and then either discards the contents or passes it up to the TCP layer. These steps were deemed necessary because diffusion is a multicast protocol, and thus we could not rule out the possibility of datagrams reaching our socket object that were not actually destined for it.

Early on in the project, it became clear that persistent connections between the various nodes was essential. A single file transfer of a shared object could result in thousands of packets traversing the network, and session setup time was simply too long over TCP and diffusion. To counteract this problem we implemented a system whereby sockets are kept open whenever possible. The first implementation of this system opened directly to a destination, and did not support multi-hop routing very well. Under this implementation, socket timeout counters were employed to close underutilized sockets. This method has inherent scalability problems, and we decided a better solution was required.

This better solution involves a multi-hop packet routing network built on top of a random graph of sensor nodes. Each node in the system has four graph parameters specified: minimum degree, maximum degree, cliquishness, and clique radius. The cliquishness parameter defines the probability of

© 2005 by Chapman & Hall/CRC

a new edge being formed to a node within the clique radius. The minimum degree and maximum degree parameters control how many neighboring nodes can exist at any point in time. The clique parameters allow us to control the size and connectedness of cliques within the graph. Cliques become more important when we investigate the index system.

To add a new edge, a random number is generated to decide whether or not to add a clique edge. Then a random node from the node cache is chosen based upon two filter criteria: the chosen node must have a minimum path length of two to this node, and its minimum path length must be less than or equal to the clique radius for a clique edge, or greater than the clique radius for a nonclique edge.

The messaging system implements the core of the REAP. At its lowest levels, this consists of a packet protocol, on top of which serialized objects are built. The 'Packet'\ class is nothing more than a variable-sized opaque data carrier that is capable of sending itself between nodes, and it also performs data and header checksumming. The header defines enough information to route packets, specify the upper level protocol, and to handle multi-packet transmissions where the number of packets is known *a priori*. The options field consists of a 4-bit options vector and a 4-bit header extension size parameter. The time-to-live (TTL) field is used in the new multi-hop protocol to eventually destroy any packet routing loops that might form.

Higher level messaging functionality is handled by a set of classes that do object serialization and by a base message class. The serialization class in REAP provides a fast method of changing common data types into network byte-ordered, opaque data. The key advantage to this serialization system is that it only handles common data types, and thus has much lower overhead than technologies such as XDR and ASN.1.

The base messaging class provides a simple interface to control destination address, source transaction information, possible system-state dependencies for message delivery, and control over sending the message. In addition, it defines abstract serialization and reordering functions that are implemented by all message types.

The serialization class sits beneath the base message class and does the physical work of serializing data, packetizing the serialized buffer, and then injecting those packets into the router. On the receiving end, packets are received by an object serialization class and inserted into the proper offset in the receive buffer. A union-find structure keeps track of packet sequence numbers and, once it detects that all packets have been received, the message is delivered to a message queue in the destination task structure.

Another interesting feature of the messaging system is the function called *run*. This function takes a task structure as an argument, and is generally intended to perform some action on the destination of the message. We will see an example of this later on when we discuss the index server.

The daemon packet router has several key responsibilities. The primary one is to use its internal routing tables to move packets from source to destination. The other primary function of the router is to coordinate the dissemination of multi-hop routing data.

The current method of determining multi-hop paths is through broadcast query messages. We gradually increase the broadcast TTL until a route is found, or a TTL upper limit is reached, at which point the node is assumed down. This methodology helps to reduce flooding, while making optimal paths likely to be found. A simple optimization allows a node to answer a multi-hop query if it has an answer in its routing table. Although this system is essentially a heuristic, it tends to work well because failed intermediate nodes are easily bypassed when their neighbors find that they cannot reach the next hop. Of course, this can lead to much longer paths through the graph, but support is integrated to warn of intermediate node failures, and multi-hop cache expire times help to reduce this problem by forcing refreshes occasionally. The multi-hop refreshes are carried out in unicast fashion, and a broadcast refresh is only used if a significant hop count increase is detected.

The actual routing of packets involves looking at the two destination fields in the packet header. First, a check is performed to determine whether the destination node identifier is equivalent to the current node's identifier, the local loopback address, or one of several addresses that are defined for special purposes, such as broadcast to all members of a clique. The next check is to determine whether the

© 2005 by Chapman & Hall/CRC

destination process identifier is equivalent to that of the current process. If it is not, then the packet will need to be forwarded across a UNIX domain socket. If both of these tests pass, then the packet must be delivered to the appropriate task. Because packets do not contain sufficient routing data to deliver them to a specific task, we must recreate the high-level message object in the router to determine the message's final destination.

Every task in a REAP process registers itself with the router during initialization. Once a task is registered, it can receive messages bound for any active ticket. Several special tickets are defined for every task that handles task status messages and for task-wide requests. Other tickets are ephemeral and are allocated as needed.

An important component of the REAP daemon is the index system. This system implements a distributed database of resource available on the network. Each record in this database describes an object of one of the following types: index server, file, executable, library, pipe, memory map, host, or a task. Every record in the database has a canonical name and resource locator associated with it. Both of these values are stored as human-readable strings. Besides this, metadata to allow for both data and metadata replication are present. The goal is to have a distributed cluster of index servers that transparently replicate each other's index records, and also to have a resource control system that transparently replicates the actual data. At this point, the replication technology is only partially implemented.

The index system consists of the following modules: client, server, database, and the associated messaging protocol. The client is responsible for building a query message, sending the message, and either waiting for a response or returning a response handle to the client in the case of an asynchronous call. The server consists of a pool of threads that poll for incoming messages on the server task structure. When a thread receives a message, it runs the query embedded in the message against the local database and then sends the results back to the client in a query result message.

The query system is based upon a fairly extensible parse tree. The query language permits complex Boolean filtering on almost any variable defined in an index record. The index server is essentially a lightweight SQL server that is tailored to resource location.

The index infrastructure is mainly built upon two message types: a query message and a result message. The query message consists of an operand tree, some query option flags, and possibly a list of index records. Once the query message reaches the server, it is received by a server thread and the run function is called. This function performs a query against the index database object and sends back a result message to the source node. Once these actions are complete, the run function returns and then the index server deallocates the query object. The index server itself is nothing more than a pool of threads that accept a certain type of message and then allow the messages to perform their actions. In this sense, the REAP messaging system implements the mobile-agent paradigm.

The other major feature of the index system is a system to select code based upon destination system architecture and operating system. To handle this, system architecture and operating system are considered polymorphic class hierarchies. Every index record contains an enumeration defining its membership in each hierarchy. When a system requests object code or binary data, we must ensure that it is compatible with the destination system. Thus, every index query can filter based upon architecture, if desired. When a query indicates that architecture and/or operating system are a concern, then C++ dynamic_cast calls are made to ensure compatibility. Because we are using the C++ dynamic casting technology, supported architectures and operating systems are determined at compile time. It would not be a technically difficult modification to use human-readable strings and run-time-defined polymorphic hierarchies. However, we chose the compile-time approach because it is faster, and the architectures and operating systems in our lab are relatively constant.

To give an example of how this technology would work, let us take an example of a sensor node having raw time series data that needs to be run through a first four year transform (FFT). Suppose a distributed process scheduler determines that it would be optimal to move the raw data to a wireless laptop that is deployed in the field. When the laptop goes to run the FFT, it queries the index database for a given FFT algorithm and requests architecture polymorphic checking. Let us say this laptop has

© 2005 by Chapman & Hall/CRC

a processor with Intel's SSE and MMX extensions, but not the SSE2 extensions. When the index server processes the query, let us say it finds FFT algorithms that are compiled for 386, Pentium, SSE, Pentium 4, and Alpha EV5. When it filters these queries, it determines that it can cast the laptop into 386, Pentium, and SSE, but not Pentium 4 or Alpha EV5. The laptop will then attempt to download the optimal one, only dropping to slower implementations when it cannot download the fastest one.

All operations in REAP are addressed by their transaction address. This address consists of the 4-tuple (node, process, task, ticket). These globally unique addresses permit flexible packet routing. A major goal of REAP is to permit network-wide interprocess communication through a simple high-level interface, without introducing high overhead. We will see how this goal is met when we discuss the resource management module of the REAP mobile-code daemon.

In order to support the complex transaction routing system, a task control structure is required. All threads and other major tasks have their own task structure. This structure is registered with the local packet router and is where the message structures get delivered. Its primary jobs are to handle message I/O and to allocate tickets. Every active ticket has an associated incoming message queue, and thus it is possible in our framework to receive messages for specific tickets. As an added feature, message type filtering is supported at the task level. Any messages which fail to pass the filter are not delivered to the task, and are instead deallocated.

Another purpose of the transaction management system is task monitoring. We employ a publish/subscribe model for this purpose. Any task may request status information from another task by subscribing to its status information service, and then every status message published by that task will be sent to the subscribed task. At the moment, all status information is sent as unicast datagrams. The main purpose of this system is to notify the requester that its request has been received, and to notify it again when the request is completed. Another interesting application of this technology could include distributed process schedulers that monitor the progress and system load on a cluster of nodes, and then schedule compute jobs to distribute the load to meet predefined criteria.

The resource management framework is tightly coupled with the index system. When a client program wants to access a resource, a query to the index system is made. The results returned can then be passed into the resource management object. The resource manager then attempts to open one of the resource from the result set. If possible, one resource from each canonical name in the result set will be opened. Thus, the resource manager is capable of overcoming node failures by looking for other copies of the same resource. The current implementation attempts to open one instance of every canonical name in parallel, and continues this iterative process as timeouts occur. Eventually, an instance of every canonical name will be opened, or the resource manager will run out of instances of a resource in the index result set.

The resource control system is built on top of a client/server framework. This framework was chosen because the types of resource we want to support are generally not concurrent objects.

Thus, the resource management system consists of two REAP message types: a resource operation message and a resource response message. Then, there are two types of resource object: a client object and a server object. For any given resource, there will exist exactly one server object and one client object per task with an open handle to the resource. When a given client wants to perform an operation on the resource it will send a resource operation message to the server object's transaction address. The server will then call the run method of the message and, through a set of polymorphic calls described below, it will perform I/O operations on the server object. A response message will then be sent to the originating node.

The client and server resource objects are based upon an abstract interface that defines several common methods that can be used on UNIX file descriptors. The major base operations are: open, close, read lock, write lock, unlock, read, write, and stat. In all cases, blocking and nonblocking versions of these functions are provided, and the blocking functions are simply built on top of the nonblocking code.

As a simple performance improvement, client and server caching objects were constructed that perform both data and metadata caching. Since our distributed resource interface is essentially identical

© 2005 by Chapman & Hall/CRC

to the virtual filesystem interface that UNIX-like kernels give to applications, standard locking semantics can apply. Thus, our caching module simply looks at the numbers of open read mode and write mode file descriptors to determine the acceptable caching strategy. For the multiple readers, and single writer cases, we allow client-side caching. For all other cases we must disable client-side caching. Thus, our caching semantics are identical to those used in the Sprite Network Filesystem [23]. The REAP framework makes our implementation very simple, because our mobile-agent-based messages can easily turn on and off client caches with minimal overhead.

To demonstrate the power of this resource control model, we have built client and server objects to support a distributed shared memory architecture. Once again, we employ the abstract client-server caching model to increase performance.

The last major component of the REAP framework is process creation and management. This portion of the architecture consists almost entirely of message types. The primary message type is a process creation message. This message contains an index record pointing to the binary to execute. It also contains the argument and environment vectors to include. A second message is a process-creation response message. This message simply contains the transaction address of the newly created process. Finally, task monitoring messages may be used to monitor the progress of a task using the publish/subscribe model discussed in Chapter 29.

The REAP mobile-code daemon permits us to experiment with many different mobile-code paradigms over a fault-tolerant multi-platform framework. Because it provides a simple cross-platform, distributed interprocess communication framework, it is very useful for developing a system of collaborating distributed processes. This approach is capable of mimicking all the major mobile-code paradigms, as shown by Orr [24]. Furthermore, its polymorphic code selection system permits us to use the optimal algorithm on a given system without significant user interaction. Finally, the distributed resource management system allows us to reduce bandwidth and permit concurrent use of resources without breaking normal concurrency rules.

40.5 Application Programming Interface

We provide here a summary of the application programming interface of the Reactive Sensor Networks Mobile Code System. It provides a high-level description of the system architecture. It describes the syntax and semantics of services, which may be of use to other projects in the Sensor IT program. These services are a subset of the full system. The initial implementation supported Windows machines using the IP. The API presented can still be modified to fit the needs of other research projects. Please notify the author of this chapter of any requests and/or comments. Nodes respond to the following API calls (semantics explained later):

status = **exec**(*program-class*, *input-data-vector*, *output-data-vector*, *resource-list*, *optional-command-line*)
status = **exec_noblock**(*program-class*, *input-data-vector*, *output-data-vector*, *resource-list*, *optional-command-line*)
status = **pipe**(*program-class*, *input-data-vector*, *output-data-vector*, *resource-list*, *optional-command-line*)
status = **kill_pipe**(*program-class*, *machine*)
status = **lock**(*program-class*, *machine*)
status = **unlock**(*program-class*, *machine*)
status = **load**(*program-class*, *machine*)
status = **register_program**(*class*, *URL*, *optional-command-line*)
status = **register_machine**(*machine*, *port*)
status = **list_machines**(*machine-info-vector*)
status = **list_classes**(*class-info-vector*)
status = **list_available_classes**(*machine*, *class-info-vector*)

© 2005 by Chapman & Hall/CRC

Parameters:

Status.	An integer value indicating call success or failure. ARL_MCN_SUCCESS indicates success. Other values indicate errors. A list of error returns will be provided with the final documentation.
program-class.	In the initial delivery this will be a string name that uniquely identifies a program.
input-data-vector.	A null-terminated array of pointers to strings giving a list of URLs indicating files to be used as input.
output-data-vector.	A null-terminated array of pointers to strings giving a list of URLs indicating files to be used as output.
resource-list.	A null-terminated array of pointers to strings of resource identifiers. In the initial implementation, this will be a node name (e.g. strange.arl.psu.edu).
machine.	A string containing a node name (e.g. strange.arl.psu.edu).
optional-command-line.	A string defining the command-line format for an executable or parameters for a DLL.
Port.	IP port number of the socket used by the ARL mobile-code software to listen for mobile code requests.
machine-info-vector.	A null-terminated array of pointers to a data structure consisting of pointers to two fields: node_name and port number.
class-info-vector.	A null-terminated array of pointers to a data structure consisting of pointers to three fields: class_name, URL, and default command line (possibly null).

API Call Semantics:

exec.	Executes in four phases: (1) uploads all data in the vector *input-data-vector* and the program *program-class* (if files are not currently located on the node); (2) executes the program; (3) writes output to the URLs in *output-data-vector*; (4) performs garbage collection. This call blocks until execution is complete. Returns the completion status of the program. The optional command-line argument can be used to override the defaults given in *register_program.*
exec_no_block.	Same as *exec*, but does not block. Returns a completion status indicating system acceptance (or nonacceptance) of the call. The optional command-line argument can be used to override the defaults given in *register_program.*
Pipe.	Executes in five phases: (1) uploads the program *program-class* (if the file is not currently located on the node) and *input-data-vector* identifies files on the node; (2) executes the program; (3) writes output to the URLs in *output-data-vector*; (4) waits for modifications to files in *input-data-vector* and then goes back to step (2); (5) performs garbage collection on receipt of the out-of-band signal from *kill_pipe*(). This call does not block. Returns a completion status indicating system acceptance (or nonacceptance) of the call. The optional command-line argument can be used to override the defaults given in *register_program.*
kill-pipe.	Sends an out-of-band message to *machine* indicating the pipeline should be terminated.
lock.	Download a program to a node. Make this program unavailable for garbage collection.
unlock.	Make a program available for garbage collection that was previously unavailable for garbage collection.
load.	If a local node attempts to execute a program not present locally, then it can use this call to trigger a network interrupt. First, nodes in the neighborhood will be signaled. If they have a copy of the program they will transfer the copy to the requesting node. Otherwise, the network interrupt will propagate

© 2005 by Chapman & Hall/CRC

through the multi-hop network up to the repository. If the program is found on a node on any of the hops, then propagation of the request will stop and the program will be transmitted to the requestor. If the program is found on the repository, then the program will be transmitted to the requestor. If the program is not found in the repository, then an error condition is signaled.

register_program. Creates a link in the repository between the URL, the default command line and the unique class name. The default command-line is either a string giving the command-line arguments used when executing the class (including variables for input and output files), or a list of default parameters used when constructing a call to a function in a DLL. In the initial delivery, class name must be unique for each program. In later releases, polymorphism and subclasses will be supported.

register-machine. Identifies a machine for use by the mobile-code software.

list_machines. Returns a pointer to a *machine-info-vector*. One entry is given for every machine registered with the mobile-code repository. This can be used to initialize a system list of nodes that the machine can accept network connections from.

list_classes. Returns a pointer to a class *info-vector*. One entry is given for each class registered with the mobile-code repository.

list_available_classes. Returns a pointer to a class *info-vector*. One entry is given for each class available for use on the node indicated by the *machine* parameter. If *machine* is null, then the local node is used as a default.

40.6 Related Work

In this section we discuss related technologies. This discussion does not include mobile-code paradigms. We discuss ongoing research that has significant overlap with our approach.

Researchers from Berkeley and Oak Ridge National Laboratories have developed a linear algebra and scientific computing software library, PHiPAC, which adapts to its underlying hardware [25,26]. The library probes the host's hardware to determine the memory and cache structure. Knowledge of the underlying substrate allows PHiPAC to modify computations and be more efficient. Performance increases of 300% have been reported. Researchers at Duke University have designed real-time task-scheduling software that varies processor clock rates for tasks with different priorities and real-time deadlines. Energy consumption has been reduced by up to 70% [27]. We consider these examples as evidence that adaptive software and hardware technologies have tremendous potentials for time and energy savings. Our work differs in that we look at applying similar concepts to mobile-code implementations. We extend the ideas of these researchers by co-adapting hardware, software and network resources.

Some mobile-code implementations may be considered adaptive. Java is probably the best-known mobile-code implementation. Applets can be downloaded and executed locally. RMI allows applets registered with a service to be executed on a remote node. Use of a standardized language allows virtual machines running on different processors to execute the same intermediate code. Our approach differs greatly from Java's ''write-once run-anywhere'' implementation, which essentially restricts the system to a single standardized language like ANSI C. Any language that can link to our API can make mobile-code calls. The mobile-code modules can be written in any language with a compiler for the host machine, or with an interpreted language with an interpreter for the host machine. Our use of daemons to run remote invocations resembles RMI to a certain extent.

Milojicic *et al.* [6] describe a number of task migration approaches. They are divided into kernel or user-space methods. Like our approach, Tui allows migration of processes among heterogeneous architectures. Tui does this by compiling programs written in a standardized language once for each

© 2005 by Chapman & Hall/CRC

target machine. Our approach supports this, but also allows programs written in different languages to cooperate. Tui does not consider code morphing for efficiency improvement. Most implementations, including MOSIX, Sprite, etc., are designed for distributed computing environments with a high-speed dependable network infrastructure. Ours uses either IP or wireless and does not assume a dependable substrate.

A natural application of mobile-code technology is network management. Halls [12] describes using the tube system to manage ATM networks efficiently. Breugst and Magedanz [28] use mobile agents to provide advanced telecommunications services, such as call screening and call forwarding. These approaches differ from our work in that they use software solely to control the hardware substrate. We do not explicitly consider software control of the underlying network.

Weiser [29] suggests the concept of ubiquitous computing. Computers, as such, should become invisible technology. Intelligence should be embedded directly into existing devices. General-purpose computers will no longer be necessary. At Xerox PARC, a number of prototypes have been constructed following this general concept. Ubiquitous computing has inspired recent research in smart spaces and commercial products like Jini [30]. Ubiquitous computing can best be achieved through cooperating networks of embedded processors.

Abelson *et al.* [31] describe a novel research agenda for a new class of computational entities. Amorphous computing is computation as carried out by a network of cooperating automata. Each individual computational entity is fallible. It is hypothesized that nanotechnology will support low-cost construction of individual low-power networked computational devices. Approaches embedding computation into living cells [32] have also been explored. Their approach to system design and evaluations uses cellular automata (CA) abstractions and is relevant to the work proposed here. They assume that all nodes have identical software and do not consider mobile code. The standard CA model is modified to assume that the grid is not regular and that communications are asynchronous. Synergy is possible between our work and amorphous computing. Their extensions to the CA model could be useful for our work. Some amorphous algorithms, such as the construction of global coordinate systems [33], are directly applicable to our work.

40.7. Summary

This chapter discusses mobile code support for sensor networks. It provides an overview of different mobile code implementations and research projects. We discussed in detail our concepts for distributed dynamic linking and the implementation of our mobile code daemons. We maintain that enabling software adaptation at this level greatly aids the ability of sensor networks to continue working in chaotic environments.

Acknowledgments and Disclaimer

This material is based on work supported by the Office of Naval Research under Award No. N00014-01-1-0859 and the Reactive Sensor Network Grant Award # F30602-99-2-0520. Any opinions, findings, and conclusions or recommendations expressed in this presentation are those of the author and do not necessarily reflect the views of the Office of Naval Research. The authors would also like to thank the anonymous referees, whose inputs have greatly improved the paper.

References

[1] Von Neumann, J., *Theory of Self-Reproducing Automata*, Burks, A.W. (ed.), University of Illinois Press, Urbana, IL, 1966.

[2] Sapaty, P., *Mobile Processing in Distributed and Open Environments*, Wiley, New York, 1999.

© 2005 by Chapman & Hall/CRC

[3] Network Forth, http://www.sandelman.ottawa.on.ca/People/Michael_richardson/network-forth.html (last accessed on 7/23/2004).
[4] Wu, D. *et al.*, StratOSphere: unification of code, data, location, scope and mobility, in *Proceedings of the International Symposium on Distributed Objects and Applications*, 1999, 12.
[5] Fuggetta, A. *et al.*, Understanding code mobility, *IEEE Transactions on Software Engineering*, 24(5), 342, 1998.
[6] Milojicic, D. *et al.* (eds), *Mobility: Processes Computers, and Agents*, Addison-Wesley, Reading, MA, 1999.
[7] Tennenhouse, D.L. *et al.*, A survey of active network research, *IEEE Communications Magazine*, 35(1), 80, 1997.
[8] Zayas, E., Attacking the process migration bottleneck, in *Proceedings of the 11th ACM Symposium on Operating Systems Principles*, November 1987, 13.
[9] Harchol-Balten, M. and Downey, A.B., Exploiting process lifetime distributions for dynamic load balancing, *ACM Transactions on Computer Systems*, 15(3), 253, 1997.
[10] Vijaykrishnan, N. *et al.*, Object-oriented architectural support for a Java processor, in *Lecture Notes in Computer Science*, vol. 1445, Springer Verlag, 1998, 330.
[11] Lynch, N.A., *Distributed Algorithms*, Morgan Kaufmann Publishers, San Francisco, CA, 1996.
[12] Halls, D.A., *Applying mobile code to distributed systems*, Ph.D. Dissertation, Computer Science, University of Cambridge, 1997.
[13] Tschudin de Bâle-ville, C.-F., On the structuring of computer communications, Ph.D. Dissertation, Informatique, Université de Genève, 1993.
[14] Jini Technology Helper Utilities and Services Specification, Sun Microsystems, Palo Alto, CA, 1999.
[15] McDowell, C.E. *et al.*, JAVACAM: trimming Java down to size, *IEEE Internet Computing*, 23, 53, 1998.
[16] Mills, K., personal communication, 1999.
[17] Brooks, R.R. *et al.*, Reactive sensor networks: mobile code support for autonomous sensor networks, in *Distributed Autonomous Robotic Systems DARS 2000*, Springer Verlag, Tokyo, October 2000, 471.
[18] Brooks, R.R., Distributed dynamic linking, Penn State Invention Declaration, May 2000.
[19] Hwang, K., *Advanced Computer Architecture*, McGraw-Hill, New York, 1993.
[20] Bender, E.A., *Mathematical Methods in Artificial Intelligence*, 1996.
[21] Moore, J. *et al.*, Tracking targets with self-organizing distributed ground sensors, in *2003 IEEE Aerospace Conference*, IEEE Computer Society, Los Amitos, CA, March 2003.
[22] Intanagonwiwat, C. *et al.*, Directed diffusion: a scalable and robust communication paradigm for sensor networks, in *Proceedings of Mobicom '00*, 2000.
[23] Nelson, M.N. *et al.*, Caching in the sprite network file system, *ACM Transactions on Computer Systems*, 6(1), 134, 1988.
[24] Orr, N., *A message-based taxonomy of mobile code for quantifying network communication*, M.S. Thesis, Penn State Computer Science and Engineering, Summer 2002.
[25] 'Self-tuning' software adapts to its environment, *Science*, 286, 35, 1999.
[26] Bilmes, J. *et al.*, The PHiPAC v1.0 matrix-multiply distribution, Technical Report TR-98-35, International Computer Science Institute, Berkeley, CA, October 1998.
[27] Swaminathan, V. and Chakrabarty, K., Real-time task scheduling for energy-aware embedded systems, *Journal of the Franklin Institute*, 338, 729, 2001.
[28] Breugst, M. and Magedanz, T., Mobile agents — enabling technology for active intelligent network implementation, *IEEE Network*, May/June, 53, 1998.
[29] Weiser, M., The computer for the 21st century, *Scientific American*, September, 94, 1991.
[30] Jini Technology Helper Utilities and Services Specification, Sun Microsystems, Palo Alto, CA, 1999.

© 2005 by Chapman & Hall/CRC

[31] Abelson, H. *et al.*, Amorphous computing, AI Memo 1666, Massachusetts Institute of Technology, Cambridge, MA, August 1999, http://www.swiss.ai.mit.edu/projects/amorphous/paperlisting.html (last accessed on 7/23/2004).

[32] Coore, D., Botanical computing: a developmental approach to generating interconnect topologies in an amorphous computer, Ph.D. Thesis, MIT Department of Electrical Engineering and Computer Science, February 1999.

[33] Nagpal, R., Organizing a global coordinate system from local information on an amorphous computer, AI Memo 1666, Massachusetts Institute of Technology, Cambridge, MA, August 1999, http://www.swiss.ai.mit.edu/projects/amorphous/paperlisting.html (last accessed on 7/23/2004).

© 2005 by Chapman & Hall/CRC

41

The Mobile-Agent Framework for Collaborative Processing in Sensor Networks*

Hairong Qi, Yingyue Xu, and Teja Phani Kuruganti

41.1 Introduction

This chapter discusses the distributed computing paradigms used to support collaborative processing in sensor networks. Sensor networks form a typical distributed environment, and the most popular computing paradigm deployed has been that of the client/server where all the clients send the raw data to a processing center for data dissemination, as illustrated in Figure 41.1(a). In some applications where the size of raw data is very large, the clients can perform some local processing and send a compressed version of the raw data, or simply the local processing results, to the processing center, as illustrated in Figure 41.1(b). This scheme is widely used in distributed detection [1]. Sometimes, the client/server-based processing can be carried out hierarchically with multiple levels of processing centers, as illustrated in Figure 41.1(c) to solve scalability problems [2].

Although popular, client/server-based distributed computing is not suitable for applications developed in sensor networks, as the sensor network possesses some unique characteristics that the client/server-based approach cannot accommodate. Here, we summarize these features as follows.

- Extremely constrained resources — limited communication bandwidth, power supply, and processing capability.
- Sheer amount of sensor nodes — a sensor network can contain up to thousands of sensor nodes.
- Fault-prone sensor nodes and communication links — due to harsh working environment and communication through unreliable wireless link.
- Exceptionally dynamic nature — existing sensor nodes can stop functioning due to power depletion, new sensors can be deployed, sensor nodes can be mobile.

*This research was supported in part by DARPA under grant N66001-001-8946.

© 2005 by Chapman & Hall/CRC

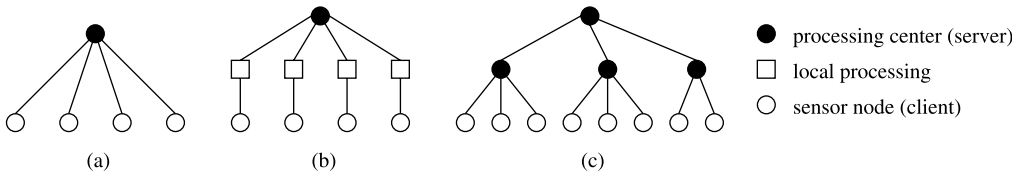


Figure 41.1. Illustration of the variants of client/server-based computing model.

These properties request the distributed computing paradigm in sensor networks to be energy efficient, scalable, fault tolerant, and adaptive.

Client/server-based computing is able to achieve scalability with the employment of the hierarchical structure. However, it cannot respond to the load changing in real time. When more sensors are deployed, it cannot perform load balancing without changing the structure of the network. The client/server-based model can also achieve fault tolerance by using redundant information from multiple sensor nodes. However, it cannot provide energy efficiency at the same time. The processing centers behave like super-nodes, which need much higher energy, storage, and computing capabilities. These will largely reduce the lifetime of the whole sensor network, especially if all sensor nodes in the network are of the same type (a homogeneous sensor network). Note that the *lifetime* of a sensor network is defined from the time the sensor network is deployed to the time the first sensor node is out of power.

In this chapter, we present the usage of the mobile-agent-based computing paradigm for collaborative processing in sensor networks. In Section 41.2 we first discuss the principles of mobile-agent-based computing and its fundamental differences from client/server-based computing. We then design and develop a mobile-agent framework (MAF) in Section 41.3. In Section 41.4 we show how the MAF is deployed in a collaborative target classification application. Finally, we summarize our discussion in Section 41.5.

41.2 Mobile-Agent-Based Distributed Computing

A mobile agent can be regarded as a special kind of software with its own attributes. Compared with traditional software, the unique features of a mobile agent are its *autonomy* and *mobility*. The mobile agent can execute autonomously [3]. Once dispatched, it can migrate from node to node performing data processing on its own, while software can typically only execute when being called upon by other routines. Mobile agents are preferred if an application requires to reduce network load, overcome network latency, and provide robust and fault-tolerant performance, etc. [4].

Although the role of mobile agents in distributed computing is still being debated mainly because of the security concern [5,6], several applications have shown clear evidence of benefiting from the use of mobile agents. For example, mobile agents are used in networked electronic trading [7], where they are dispatched by the buyer to the various suppliers to negotiate orders and deliveries, and then return to the buyer with their best deals for approval. Instead of having the buyer contact the suppliers, the mobile agents behave like representatives, interacting with other representatives on the buyer's behalf, and alert the buyer when something happens in the network that is important to the buyer. Another successful example of using mobile agents is distributed information retrieval and information dissemination [8–11]. Agents are dispatched to heterogeneous and geographically distributed databases to retrieve information and return the query results to the end-users. Mobile agents are also used to realize network awareness [12] and global awareness [13]. Network-robust applications are of great interest in military situations today. Mobile agents are used to be aware of and reactive to the continuously changing network conditions to guarantee successful performance of the application tasks.

Mobile-agent-based computing was first proposed to support collaborative processing in sensor networks by Qi and co-workers [14,15]. Since then, a series of developments has taken place to improve the design. Related studies include performance evaluation using simulation tools of distributed

© 2005 by Chapman & Hall/CRC

computing paradigms in sensor networks [16,17], and successful applications of mobile agents in ground-target classification [18,19].

41.2.1 Mobile-Agent Attributes and Life Cycle

We consider the mobile agent as an entity with four attributes: identification, itinerary, processing code, and storage. The *identification* uniquely identifies the mobile agent. The *itinerary* specifies the route of agent migration; it can be predefined or derived adaptive to the change of network status. The *processing code* is the executable carried by the agent and performed at each local sensor node; this code is task adaptive, which enables the network to perform different tasks according to the processing code. The *storage* is used to save partially integrated results from processing done at previously migrated nodes. Upon arriving at a local node, the mobile agent is able to start the processing code, terminate its execution, record the execution status, select the next stop, and resume processing at the next node, all autonomously.

To clarify the different states where the mobile agent might reside, Figure 41.2 uses a finite-state machine (FSM) to describe the mobile-agent life cycle. Upon *creation*, the mobile agent carries the processing code and *migrates* from node to node following the itinerary. Upon arriving at each sensor node, the mobile agent *executes* the processing code and the result is saved in the storage. If the node is the last node in the itinerary or if the accuracy of the result satisfies the requirement of a specific task, then the agent returns to its dispatcher and *terminates* itself; otherwise, the agent continues the migration and execution process.

From the above discussion, we can identify two fundamental differences between the mobile-agent-based model and the client/server-based model: what is transferred over the network (agents with processing code versus data) and where the processing takes place (local nodes versus processing center). With these unique features, mobile-agent-based processing provides a few important benefits in supporting collaborative processing in sensor networks. Besides scalability and fault tolerance, which the client/server-based model also possesses, the mobile-agent-based model presents the following features:

- *Reliability*. Mobile agents can justify their itinerary such that they can always be sent when the network connection is alive and return results when the connection is re-established. Therefore, the performance of mobile-agent-based computing is not affected much by the reliability of the network.
- *Task adaptivity*. Mobile agents can be programmed to carry different task-specific processing codes, which extend the functionality of the network.
- *Progressive accuracy*. A mobile agent always carries a partially integrated result, generated by nodes it has already visited. As the mobile agent migrates from node to node, the accuracy of the integrated result is constantly improved, assuming the agent follows the path determined based on the information gain. Therefore, the agent can return results and terminate its itinerary at any time when the integration accuracy satisfies the requirement. This feature, on the other hand, also saves network bandwidth and computation time, since unnecessary node visits and data transfers are avoided.

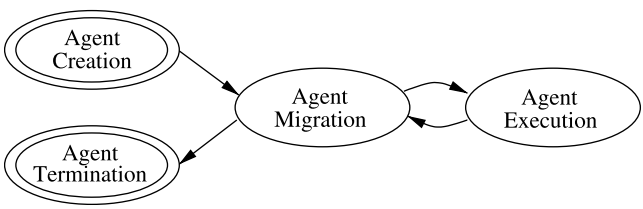


Figure 41.2. The mobile-agent lifetime, an FSM illustration.

© 2005 by Chapman & Hall/CRC

- *Balance between energy awareness and fault tolerance*. This is the most important feature of mobile-agent-based computing. On the one hand, redundancy is employed to tolerate potential faults, i.e. the mobile agent needs to integrate local results generated by multiple sensor nodes. On the other hand, the mobile agent can justify its itinerary on-the-fly, such that unnecessary node visits are avoided (as mentioned above), and hence saving both transmission and receiving energy, as well as computation energy.

41.2.2 Performance Evaluation

Although the mobile-agent-based model possesses many advantages over the client/server-based model, it does not always perform better, as the mobile-agent creation and dispatch also bring overheads. The performance of different computing models also depends on many other parameters related to the network configuration.

We designed two metrics to help evaluate these two computing models. The *execution time* is defined from the time a task is initiated by the user until the time the user obtains the result. The *energy consumption* is the amount of energy consumed during the entire execution time.

We use the Network Simulator 2 (*ns*-2) [20] simulation software to simulate the sensor networks. *ns*-2 is a discrete event simulator targeted at networking research. It is the most popular choice of simulator used in academia. To simplify the simulation, we assume that no events occur simultaneously in the sensor field and that if an event does occur, then all sensor nodes can detect it and collect the raw data. We choose a sensor field of $10 \times 10\,\mathrm{m}^2$, a random node deployment model, a random waypoint model for node mobility (i.e. the node chooses a random destination and moves at a speed of 10 m/s), a network transmission rate of 2 Mbps, and a data processing rate of 100 Mbps. Interested readers are referred to Xu and Qi [36] for detailed discussion about the simulation setup. Here, we present the simulation results when changing one of the following four parameters:

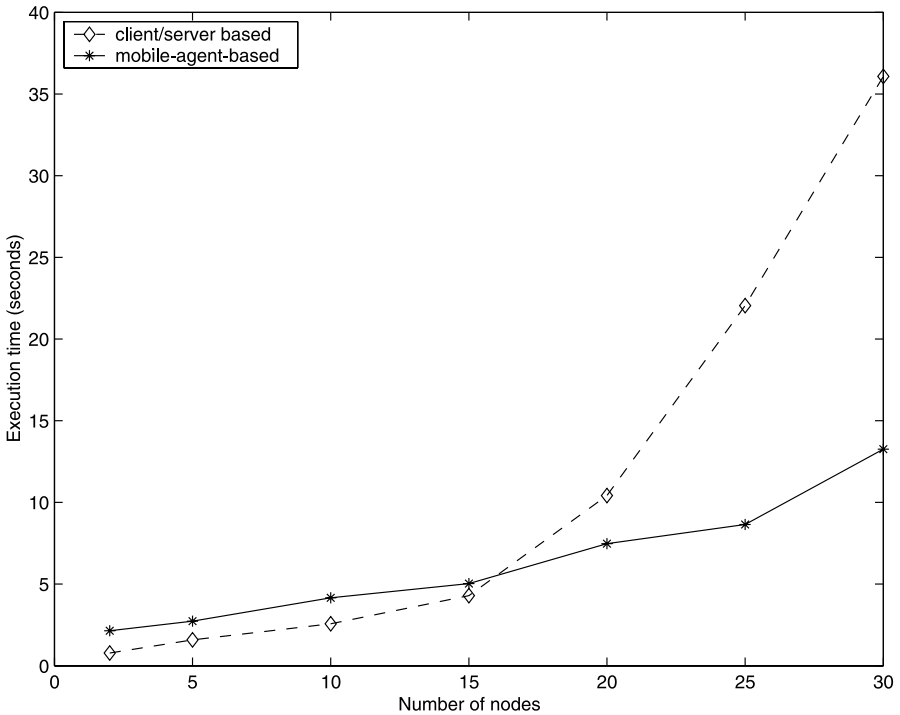
- The number of nodes in the network.
- The number of mobile agents deployed.
- The overhead ratio between mobile-agent computing and client/server-based computing ($r_{\mathrm{oh}} = o_{\mathrm{f}}/o_{\mathrm{a}}$), where the overhead of mobile-agent computing o_{a} comes from the agent creation, dispatch and receiving, and the overhead of client/server-based computing o_{f} comes from the time spent on large file access.
- The ratio between the file size s_{f} and the mobile agent size s_{a} ($r_{\mathrm{size}} = s_{\mathrm{f}}/s_{\mathrm{a}}$).

Figures 41.3–41.5 show the performance profiles of the two metrics when we change one of the four parameters listed above while keeping the others unchanged.

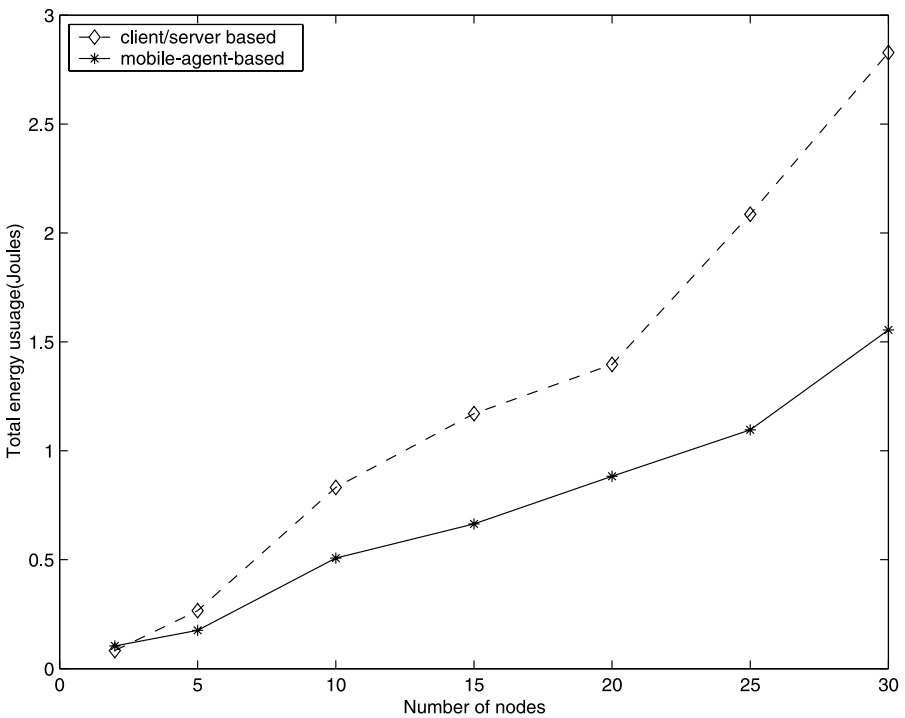
In Figure 41.3 we change the number of nodes in the network from 2 to 30 and dispatch just one mobile agent. The overhead ratio is $r_{\mathrm{oh}} = 1{:}4$ and the size ratio is $r_{\mathrm{size}} = 10{:}1$. We observe that although both profiles in Figure 41.3(a) and (b) grow as the number of nodes increases, the profiles from the mobile-agent-based approach grows much more slowly than those from the client/server-based approach, indicating the better scalability of the mobile-agent model, especially when the number of nodes is large. When the number of nodes is small (less than 15), the execution time of the client/server-based model is less than that of the mobile-agent model. This happens when the mobile agent overhead is still a factor compared with the benefit it brings. However, when the node number goes beyond 15, the mobile-agent model starts to show advantages over the client/server model. A similar pattern can be observed from the energy consumption comparison, except that the mobile-agent approach shows advantages when the number of sensor nodes is still small.

Figure 41.4 shows the profiles of the two metrics when we fix the number of nodes at 15, use one mobile agents, r_{oh} at 1:4, but change the r_{size} by keeping the mobile agent size at 1 Kb and changing the file size from 1 Kb to 50 Kb, i.e. changing r_{size} from 1 to 50. We expect to see a flat profile for both metrics for the mobile-agent-based model. From Figure 41.2(a), we observe that when the transferred

© 2005 by Chapman & Hall/CRC



(a)



(b)

Figure 41.3. The effect of the number of nodes: (a) execution time; (b) energy consumption (© 2003 IEEE) [17].

© 2005 by Chapman & Hall/CRC

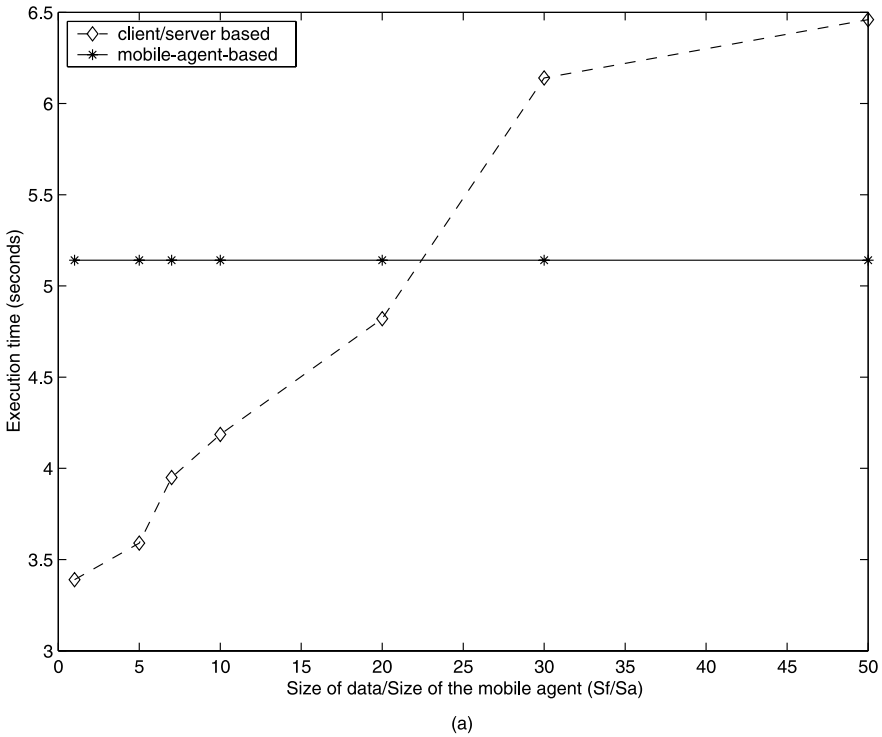
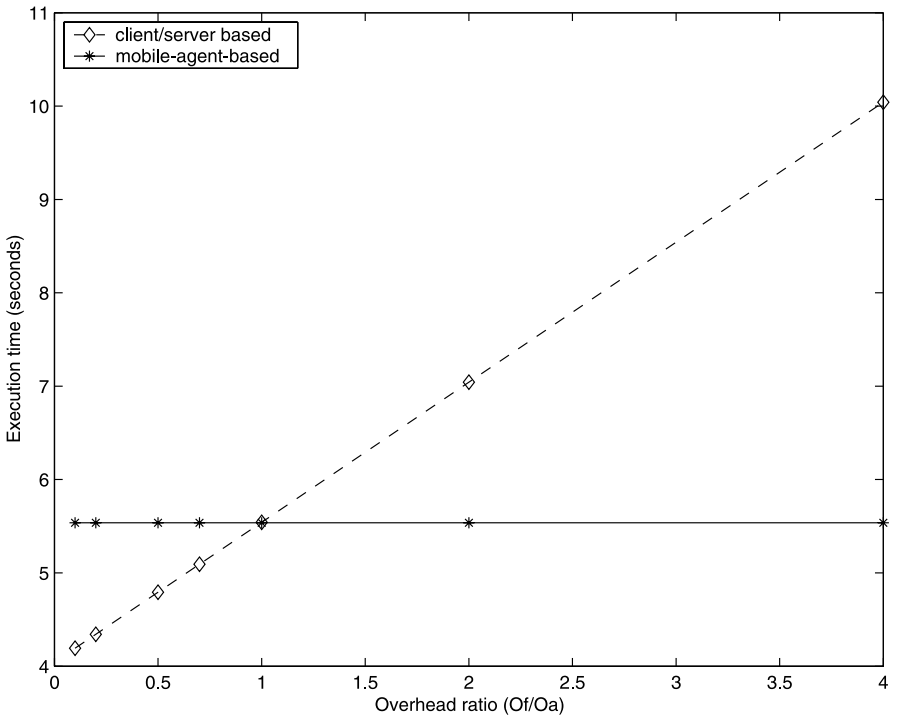
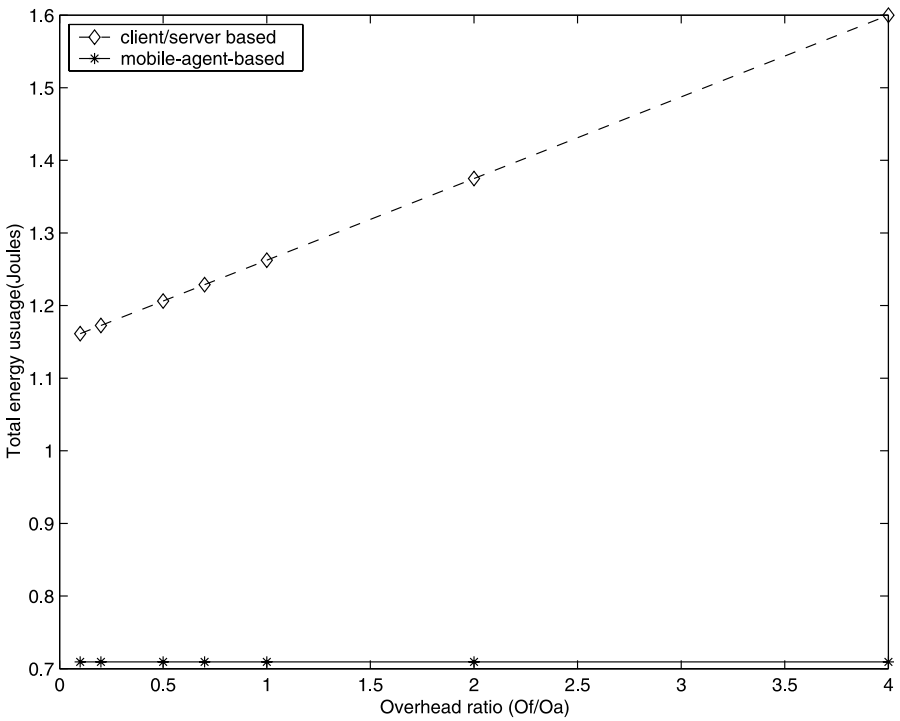


Figure 41.4. The effect of data size versus mobile agent size: (a) execution time; (b) energy consumption (© 2003 IEEE) [17].

© 2005 by Chapman & Hall/CRC



(a)



(b)

Figure 41.5. The effect of the overhead ratio: (a) execution time; (b) energy consumption (© 2003 IEEE) [17].

© 2005 by Chapman & Hall/CRC

data size goes beyond 22 Kb (i.e. $r_{size} > 22$), the client/server-based approach starts to perform worse than the mobile-agent approach in terms of the execution time. A similar pattern occurs to the energy consumption, except that the turning point is $r_{size} = 4{:}1$. This comparison indicates how small the file size needs to be in order for the client/server-based model to perform better than the mobile-agent model.

The third experiment evaluates the effect of the overhead ratio r_{oh}. We again fix the number of nodes at 15 and use one mobile agent. We keep r_{size} at 10:1, but change r_{oh} from 0.1 to 4.0. We do this by keeping the overhead of the mobile agent a constant and only change the overhead for file access. As shown in Figure 41.5, when $r_{oh} > 1$, the mobile-agent model performs better, while the energy consumption always indicates a preference over the mobile-agent paradigm even when $r_{oh} < 1$.

In the last experiment, we fix the number of sensor nodes at 100, r_{oh} at 1:4 and r_{size} at 10:1, but we change the number of mobile agents dispatched from 1 to 50. The objective is to observe the effect of the number of agents in a densely deployed sensor field. From Figure 41.6, besides the obvious profile pattern we expect, i.e. a constant profile from the client/server model and the always better performance of the agent model, we also observe an interesting phenomenon in Figure 41.6(a) when the number of mobile agents equals five. This is the place where the mobile agent spends the least amount of time for execution. The reason for this is that in order to handle the processing among 100 sensor nodes, more mobile agents should be dispatched to provide scalability. Therefore, the more agents, the less the execution time. However, while the number of agents increases, the overhead time spent on agent creation, dispatch, and receiving also increases. When the number of agent reaches six, the overhead surpasses the benefits of the mobile agent and the execution time starts to increase. However, this growth pattern is not shown in the energy consumption profile.

The aforementioned simulations aim at helping the reader understand that mobile-agent-based computing does not always perform better than client/server-based computing and that different computing paradigms are most appropriate only for a certain network setup. While client/server-based

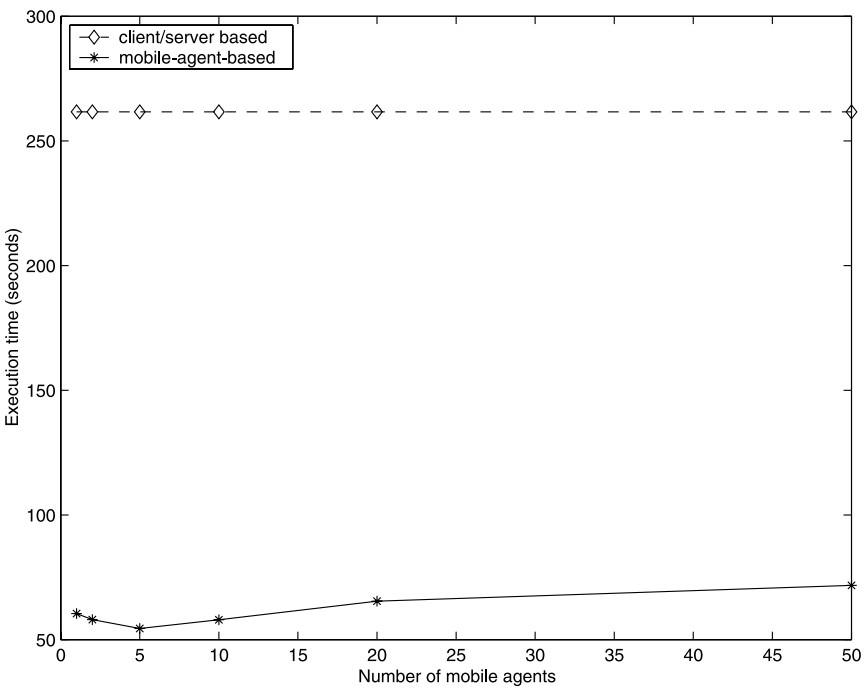


Figure 41.6. The effect of the number of mobile agents: (a) execution time; (b) energy consumption (© 2003 IEEE) [17].

© 2005 by Chapman & Hall/CRC

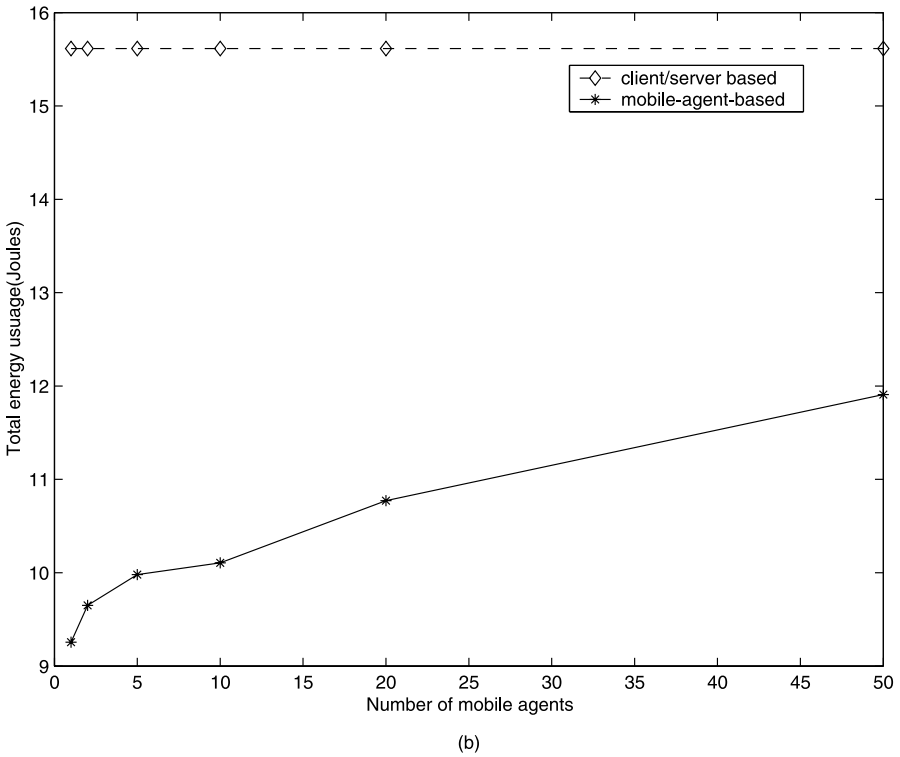


Figure 41.6. Continued.

computing works better when there are only several nodes in the network or the raw data size is not large, mobile-agent-based computing is more suitable for networks with a lot of sensor nodes. By carefully choosing the number of mobile agents dispatched, we can minimize the execution time of a task, as well as reduce the energy consumption of the network.

41.3 The MAF

We design and develop an MAF to support collaborative processing in the context of sensor networks. The system architecture is shown in Figure 41.7. Different layers in this architecture perform different tasks and provide upward layer services. Compared with a traditional seven-layered network architecture, this is a unique integrated cross-layer design that is application oriented and data-centric [21].

The collaborative processing layer hosts algorithms for the integration of information derived from multiple sensors. The MAF layer provides the mobile-agent-based computing paradigm to achieve the collaborative processing task. The MAF is realized using some routing protocols which facilitate communication over a wireless link. Note that, as shown in Figure 41.7, collaborative processing does

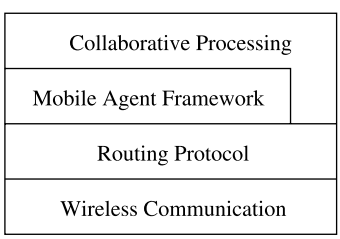


Figure 41.7. Architectural overview of the system.

© 2005 by Chapman & Hall/CRC

not have to utilize support from the MAF, it can bypass the MAF and use an application-oriented routing protocol like directed diffusion [21] directly.

There have been many mobile-agent systems developed recently. Most of them use Java or a combination of C/C++ and a scripting language, such as IBM's Aglets [22], Dartmouth's Agent Tcl [23], General Magic's Telescript [24], etc. The MAF is implemented purely in Python and has left the flexible interface to other processing modules. Lutz and Ascher [25] provide a detailed list of the benefits of Python. The article by Raymond [26] is also a good resource. We summarize the benefits as follows:

- It is an object-oriented language from the ground up. Python is ideal as a scripting tool for object-oriented languages like C++ and Java. It can easily *glue* components written in these languages.
- It is free and very well supported. Python comes with all popular Linux distributions, such as Debian, Caldera, Redhat, etc.
- It is portable. Python is written in portable ANSI C, and compiles and runs on virtually every major platform in use today [25], including Linux, Unix, and Windows.
- It supports object serialization, which is one way to save information between program executions. Object serialization is done by the Python `pickle` module, a standard function of the Python system. It converts Python in-memory objects to and from a single linear string format, suitable for shipping across network sockets, etc. [27]. The whole process is transparent to the end-user. It needs to be clarified that object serialization only keeps the data space, not the execution status. No modification to the Python interpreter is needed in order to support this kind of *moderate mobility*.
- It is simple and suitable for rapid prototyping. Python is designed to optimize development speed. It is very efficient in proof-of-concept implementation.

In order to make use of the existing, mature integration modules, the MAF also provides a flexible interface where the integration modules developed in C/C++ can be dynamically linked and executed by the agents.

The two interfaces between the application routines (C/C++) and MAF services (Python) and between the routing routines (C/C++) and MAF services (Python) are implemented by generating shared library modules accessible both by the Python and C/C++ code. These shared libraries are generated using a code development tool called Simplified Wrapper and Interface Generator (SWIG) [28]. SWIG is an open-source software development tool that connects programs written in C and C++ with common scripting languages such as Python and Tcl/Tk.

The implementation interfaces are illustrated in Figure 41.8. We have also created a Website at Source Forge [15] with a complete implementation of the MAF in Python.

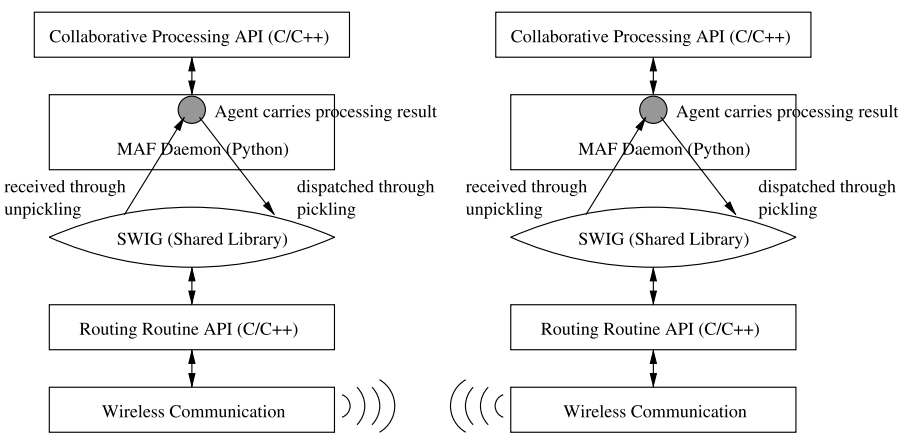


Figure 41.8. MAF implementation interfaces.

© 2005 by Chapman & Hall/CRC

41.4 Application Examples

In this section we take collaborative target classification as an example to show how the mobile-agent-based computing model can support collaborative processing. In order to use mobile-agent-based computing, there are a few components need to be realized first:

- The format of the local processing result.
- An integration algorithm that can fuse local processing results from node to node.
- An algorithm to determine the mobile-agent itinerary.

41.4.1 Format of the Local Processing Result

A reasonable choice of the format for the local result is a *confidence range*. This indicates how confident the local node is about the local processing result based on data collected from the local sensors. For example, in the application example of target classification, the confidence range is the range of classification confidence over each possible target class. The sensor output might be expressed as "I am 40 to 70% sure that the target just went by me is a diesel truck, 20 to 30% sure that the target is an SUV." The confidence range can then be represented as a matrix using the lowest belief and the highest belief for each possible target class. In this case, the confidence matrix is

$$\begin{bmatrix} 0.4 & 0.7 \\ 0.2 & 0.3 \end{bmatrix}$$

where the first row in the matrix indicates the confidence range for target class "diesel truck" and the second row indicates the confidence range for "SUV."

The *confidence* itself can be modeled by different stochastic distributions, the simplest of which would be a uniform distribution, where equal weight has been put on each confidence value within the confidence range. Other appropriate distributions could be a Gaussian (more weight on the central confidence within the confidence range) or a Rayleigh (more weight on the low confidence within the confidence range), as shown in Figure 41.9.

A one-dimensional (1-D) array can serve as an appropriate data structure to represent the confidence range. Different resolutions of processing determine the size of the array. We always assume that the confidence ranges between 0 and 1. If the processing resolution is 0.05 and there are three possible targets, then the size of the 1-D array is $(1/0.05 + 1) \times 3 = 63$ units of floating points.

41.4.2 The Integration Algorithm

The integration algorithm needs to be both effective and simple for purposes of energy efficiency and real-time response. Interested readers are referred to Qi *et al.* [29] for a comparison of different integration algorithms. A good algorithm is the *overlap function*, first proposed by Prasad *et al.* [30]. It is similar to a histogram function, where, according to the confidence range generated from multiple sensors, the overlap function *accumulates* the number of sensors with the same confidence value. Figure 41.10 illustrates the construction of an overlap function for a set of six sensor nodes when the uniform and Gaussian distributions are used.

The integrated information lies within regions over which the maximal peaks of the overlap function occur with the largest spread. The original proposal for generating and analyzing the overlap function is centralized and can only take place at the processing center. A distributed integration of the overlap function is proposed by Qi *et al.* [29] for mobile-agent-based computing such that, at each stop of the itinerary, a *partially integrated* result can be generated from previously migrated sensor node outputs and the accuracy of the information derived from this result can be used to justify whether the mobile agent needs to continue with its migration. The distributed method would generate the same output as the centralized method if the mobile agent finishes its itinerary.

© 2005 by Chapman & Hall/CRC

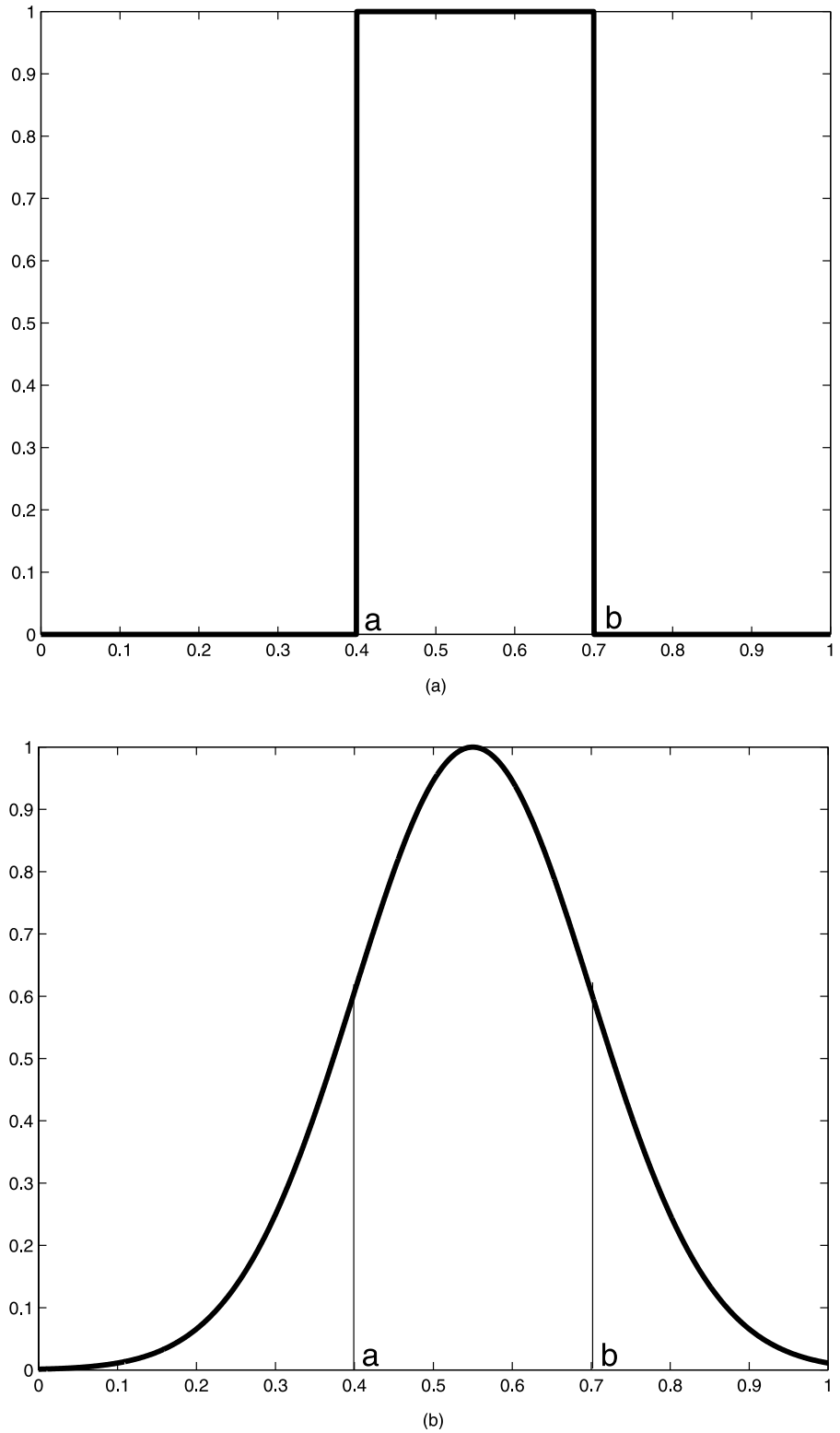


Figure 41.9. Possible choices for modeling the distribution of the confidence within a certain confidence range $[a, b] = [0.4, 0.7]$: (a) uniform; (b) Gaussian; (c) Rayleigh (next page).

© 2005 by Chapman & Hall/CRC

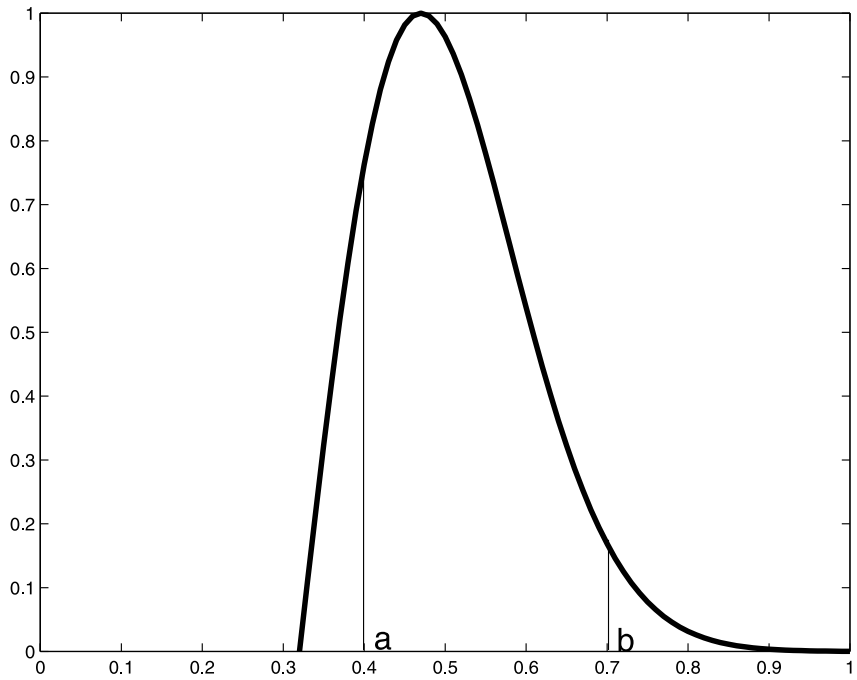


Figure 41.9. Continued.

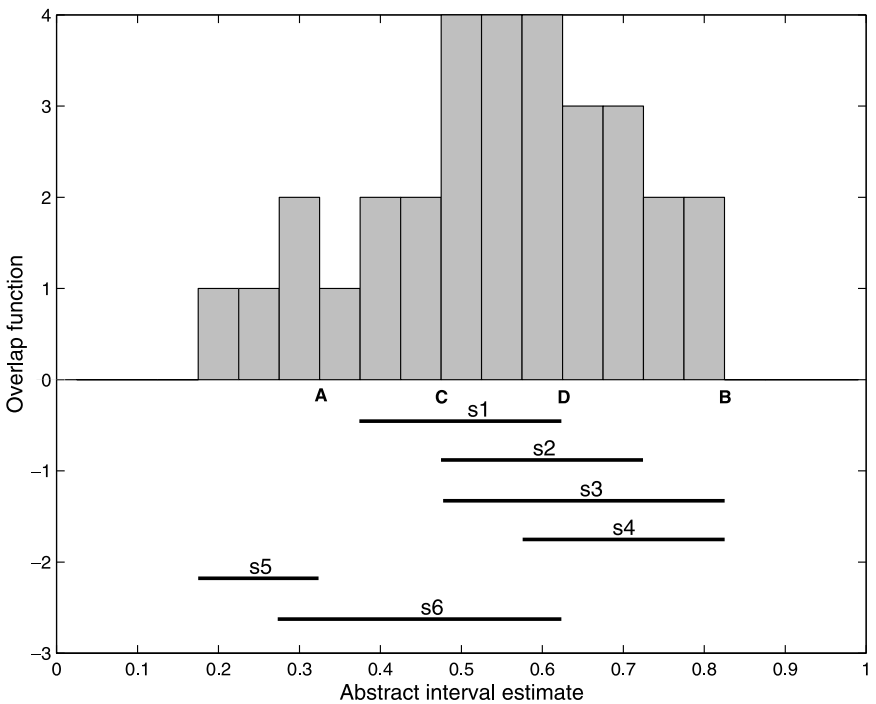


Figure 41.10. The overlap function for a set of six sensors using two distribution models (resolution is 0.05): (a) uniform; (b) Gaussian (next page).

© 2005 by Chapman & Hall/CRC

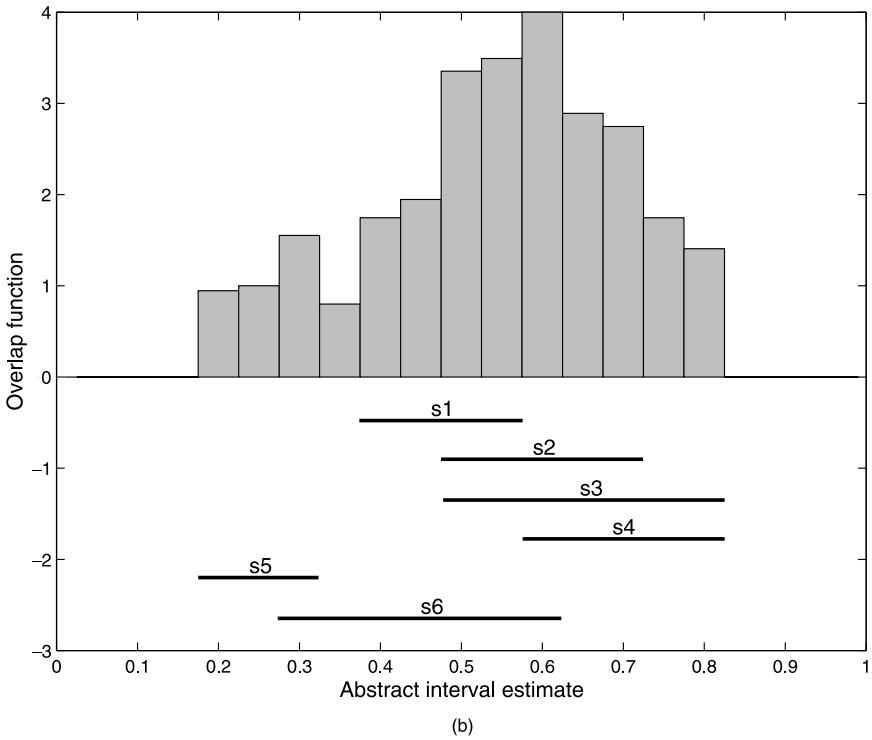


Figure 41.10. Continued.

Qi *et al.* [18] proposed a protocol in order to decide whether or not the mobile agent should stop its migration. The protocol includes the following three criteria:

1. The overlap function has its highest peaks ranging from $[n-f, n]$, where n is the number of nodes and $f = \lfloor (n-1)/3 \rfloor$ is the maximum number of faulty sensor nodes that the sensor network can tolerate. This equation comes from the Byzantine generals problem [31], which specifies the number of faults f that a certain number of sensor nodes n can tolerate.
2. The accuracy (multiplication between the height of the highest peak in the overlap function, the width or spread of the peak, and the confidence value at the center of the peak) calculated from the intermediate integration result at each sensor node has to be equal to or greater than 0.5.
3. Both (1) and (2) have to be satisfied in two adjacent migrations, excluding the first sensor node.

The itinerary of the mobile agent is critical in saving energy and achieving the required accuracy efficiently. Take the overlap function shown in Figure 41.10(a) as an example: a different itinerary generates different partially integrated results. Figure 41.11 shows the step-by-step integration result as the mobile agent migrates among nodes s_1 to s_6 where s_4 and s_5 are faulty. In Figure 41.11(a), the mobile-agent itinerary is from s_1 to s_6 in sequence. We can see that the mobile agent can actually terminate the migration after the third stop (s_3) when all the criteria have been satisfied. On the other hand, in Figure 41.11(b), we intentionally change the itinerary, and hence the order of integration. We observe that the agent can stop only after the fourth integration.

41.4.3 Mobile-Agent Itinerary

As shown in Figure 41.11, the mobile-agent itinerary is critical in providing an energy-efficient and reliable solution for collaborative processing in sensor networks.

© 2005 by Chapman & Hall/CRC

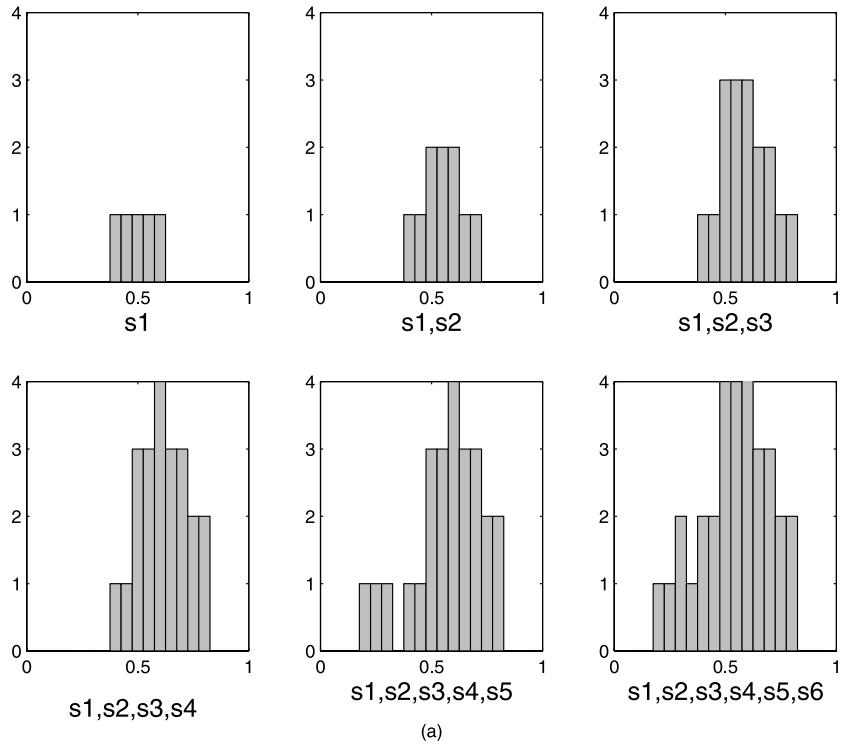


Figure 41.11. Partially integrated overlap function generated by the mobile agent with different integration order: (a) s_1, s_2, s_3, s_4, s_5, s_6; (b) s_4, s_5, s_1, s_2, s_3, s_6.

© 2005 by Chapman & Hall/CRC

The problem of determining the mobile-agent itinerary can be traced back to the famous travelling salesman problem (TSP) [32], where an *optimal* path (the shortest path in this context) is pursued for a salesman to travel through a set of cities. Although the problem is easy to solve when the number of cities is small, it becomes NP-complete when the number of cities is large, i.e. the algorithm shows an exponential growth of run time with a growing number of cities [33]. The travelling agent problem (TAP) is discussed by Moizumi and Cybenko [34] for determining the mobile-agent itinerary in applications like information retrieval from the Internet. The ith site carries a probability of p_i of successfully providing the information needed by the mobile agent. Moizumi and Cybenko [34] show that, if the latencies between any two nodes are constant, an optimal path can be obtained in polynomial time.

Both the TSP and the TAP are global optimization problems where a centralized processing environment is anticipated. However, this is not the case in sensor networks, and is especially not suitable for mobile-agent-based distributed computing. PARC developed an information-driven dynamic sensor collaboration for a target tracking application [35,36]. In order to achieve energy-efficient computing, it selects the next node which most likely improves the tracking accuracy based on both the information constraints and constraints on cost and resource consumption. Specifically, the approach formulates the tracking problem as a sequential Bayesian estimation problem. In order to estimate the track, each new sensor measurement is combined with the current estimate to improve the estimation accuracy. Therefore, the problem of selecting the next sensor can be formulated as an optimization problem as well. However, this optimization problem only requires local information. The design of the mobile-agent itinerary follows a very similar idea.

We assume the sensor nodes exchange information among neighbors periodically or when there is a dramatic change in the information content. We design the content of the information exchange to include three pieces of information: the signal energy sensed, the remaining energy onboard the sensor node, and the location of the sensor node. For example, if there is a target passing by the sensor node, then the signal energy sensed by the sensor could increase by a noticeable amount, or if the sensor node just moved from one site to another, then the location of the node would change as well. When the agent arrives at a local node it uses the information provided from the neighboring nodes to find out its next hop, which should possess as high a signal energy as possible, as much remaining power as possible, and as close a distance to the current node as possible. This idea was partly implemented in Qi *et al.* [37].

An even simpler itinerary determination method is for the agent to choose its next hop randomly among neighboring nodes. An evaluation study is under way to compare the performance between random selection and optimal selection either using local information or global information. Since the sensor network is such a dynamic environment, we expect the random approach to present reasonably good results in adapting to changes of network topology.

41.5 Summary

This chapter discussed two distributed computing paradigms used to support collaborative processing in sensor networks. We compared the performances of client/server-based computing and mobile-agent-based computing. We showed through simulation that, if the network has a large number of nodes and the size of the data file transferred is large, which is typical of setups in sensor networks, the mobile-agent-based model is advantageous compared with the client/server-based model. We described the implementation of the MAF and its interfacing architecture. Finally, we addressed three components in carrying out mobile-agent-based processing: the format of the local processing result, the integration algorithm, and the algorithm for determining the mobile-agent itinerary. We showed that the agent itinerary is critical in providing both energy-efficient and fault-tolerant solutions. The mobile-agent-based model also helps realize adaptivity to both task and network topology changes.

© 2005 by Chapman & Hall/CRC

References

[1] Viswanathan, R. and Varshney, P.K., Distributed detection with multiple sensors: part I — fundamentals, *Proceedings of the IEEE*, 85(1), 54, 1997.

[2] Ivy — a sensor network infrastructure for the College of Engineering, http://www-bsac.eecs.berkeley.edu/projects/ivy.

[3] Franklin, S. and Graesser, A., Is it an agent, or just a program? A taxonomy for autonomous agents, in *Third International Workshop on Agent Theories, Architectures, and Languages*, Carbonell, J.G. and Siekmann, J. (eds), volume 1193, Lecture notes in Computer Science (LNCI), Springer-Verlag, 1996, http://www.msci.memphis.edu/~franklin/AgentProg.html.

[4] Lange, D.B. and Oshima, M., Seven good reasons for mobile agents, *Communications of the ACM*, 42(3), 88, 1999.

[5] Harrison, C.G. *et al.*, Mobile agents: are they a good idea? Technical Report RC 19887, IBM Thomas J. Watson Research Center, March 1995, http://www.research.ibm.com/massive/mobag.ps (last accessed on 8/5/2004).

[6] Milojicic, D., Trend wars — mobile agent applications, *IEEE Concurrency*, 7(3), 80, 1999.

[7] Dasgupta, P. *et al.*, Magnet: mobile agents for networked electronic trading, *IEEE Transactions on Knowledge and Data Engineering*, 11(4), 509, 1999.

[8] Hattori, M. *et al.*, Agent-based driver's information assistance system. *New Generation Computing*, 17(4), 359, 1999.

[9] Kay, J. *et al.*, ATL postmaster: a system for agent collaboration and information dissemination, in *Proceedings of the 2nd International Conference on Autonomous Agents*, Minneapolis, MN, 1998, 338.

[10] Oates, T. *et al.*, Cooperative information-gathering: a distributed problem-solving approach, *IEE Proceedings — Software Engineering*, 144(1), 72, 1997.

[11] Wong, J.S. and Mikler, A.R., Intelligent mobile agents in large distributed autonomous cooperative systems, *Journal of Systems and Software*, 47(2), 75, 1999.

[12] Caripe, W. *et al.*, Network awareness and mobile agent systems, *IEEE Communications Magazine*, 36(7), 44, 1998.

[13] Ross, K.N. *et al.*, Mobile agents in adaptive hierarchical Bayesian networks for global awareness, in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, 1998, 2207.

[14] Qi, H. *et al.*, Multi-resolution data integration using mobile agents in distributed sensor networks, *IEEE Transactions on Systems, Man, and Cybernetics C*, 31(3), 383, 2001.

[15] Qi, H. and Wang, F., Mobile agent framework, http://maf.sourceforge.net, 2000 (last accessed on 8/5/2004).

[16] Xu, Y. and Qi, H., Performance evaluation of distributed computing paradigms in mobile *ad hoc* sensor networks, in *The 9th International Conference on Parallel and Distributed Systems (ICPADS)*, Taiwan, December 2002, IEEE, 451.

[17] Xu, Y. *et al.*, Mobile-agent-based computing model for collaborative processing in sensor networks, in *Global Telecommunications Conference (GLOBECOM)*, Volume 6, Los Angeles, CA, December 2003, 3531.

[18] Qi, H. *et al.*, Mobile-agent-based collaborative processing in sensor networks, *Proceedings of the IEEE*, 91(8), 1172, 2003.

[19] Wang, X. *et al.*, Collaborative multi-modality target classification in distributed sensor networks, in *Proceedings of the Fifth International Conference on Information Fusion*, volume 2, Annapolis, MA, July 2002, 285.

[20] LBL, Xerox PARC, UCB, and USC/ISI, The Network Simulator, ns-2, <http://www.isi.edu/nsnam/ns> (last accessed on 8/5/2004).

[21] Estrin, D. *et al.*, Next century challenges: scalable coordination in sensor networks, in *International Conference on Mobile Computing and Networking (MobiCom)*, Seattle, WA, August 1999, 263.

[22] Aglets. http://www.trl.ibm.com/aglets/ (last accessed on 8/5/2004).

© 2005 by Chapman & Hall/CRC

[23] Agent TCL, http://agent.cs.dartmouth.edu/ (last accessed on 8/5/2004).
[24] Telescript, http://www.science.gmu.edu/~mchacko/telescript/docs/telescript.html (last accessed on 8/5/2004).
[25] Lutz, M. and Ascher, D., *Learning Python*, O'Reilly, 1999.
[26] Raymond, E.S., Why Python? *Linux Journal*, May, 73, 2000, http://www2.linuxjournal.com/article.php?sid=3882 (last accessed on 8/5/2004).
[27] Lutz, M., *Programming Python*, 2nd ed., O'Reilly, 2001.
[28] SWIG, http://www.swig.org (last accessed on 8/5/2004).
[29] Qi, H. *et al.*, Distributed sensor networks — a review of recent research, *Journal of the Franklin Institute*, 338, 655, 2001.
[30] Prasad, L. *et al.*, Fault-tolerant sensor integration using multiresolution decomposition, *Physical Review E*, 49(4), 3452, 1994.
[31] Lamport, L. *et al.*, The Byzantine generals problem, *ACM Transactions of Programming*, 4(3), 382, 1982.
[32] Robinson, J.B., On the Hamiltonian game (a traveling-salesman problem). *RAND Research Memorandum*, 1949, RM-303.
[33] Dantzig, G. *et al.*, Solution of a large-scale traveling-salesman problem, *Operations Research*, 2, 393, 1954.
[34] Moizumi, K. and Cybenko, G., The travelling agent problem. Technical report, Dartmouth College, Hanover, NH, February 1998.
[35] Chu, M. *et al.*, Scalable information-driven sensor querying and routing for *ad hoc* heterogenous sensor networks, in *International Journal of High Performance Computing Applications*, 2002.
[36] Zhao, F. *et al.*, Information-driven dynamic sensor collaboration for tracking applications, *IEEE Signal Processing Magazine*, 19(2), 61, 2002.
[37] Qi, H. *et al.*, Distributed multi-resolution data integration using mobile agents, in *IEEE Aerospace Conference*, Volume 3, Big Sky, MT, March 2001, IEEE, 1133.

© 2005 by Chapman & Hall/CRC

42
Distributed Services

Alvin S. Lim

42.1 Introduction

Distributed sensor systems are useful for gathering critical and real-time information from many dispersed integrated low-powered sensors and mobile devices [1–3] that could steer the control operations of dynamically changing enterprise systems, such as the battlefield and manufacturing and commercial inventory and distribution systems. These mobile and miniaturized information devices are equipped with embedded processors, wireless communication circuitry, information storage capability, smart sensors, and actuators. These sensor nodes networked in an *ad hoc* way, with little or no fixed network support, to provide the surveillance, targeting, and feedback information for dynamic control of enterprises. Sensor devices are mobile, subject to failure, deployed spontaneously, and repositioned for more accurate surveillance. Despite these dynamic changes in configuration of the sensor network, critical real-time information must still be disseminated dynamically from mobile sensor data sources through a self-organizing network infrastructure to the components that control dynamic replanning and reoptimization of the theater of operation based on newly available information.

Since a large number of sensor devices may need to be quickly and flexibly deployed in impromptu networks, each sensor device must be autonomous and capable of organizing itself in the overall community of sensors to perform coordinated activities with global objectives. When spontaneously placed together in an environment, these sensor nodes should immediately know about the capabilities and functions of other sensor nodes and work together as a community system to perform cooperative tasks and networking functionalities. Sensor networks need to be self-organizing, since they are often formed spontaneously from large numbers of mixed types of node and may undergo frequent configuration changes. Some sensor nodes may provide networking and system services and resources to other sensor nodes. Others may detect the presence of these nodes and request services from them.

42.2 Purposes and Benefits of Distributed Services

Distributed services are necessary for enabling sensor nodes to self-organize into impromptu networks that are incrementally extensible and dynamically adaptable to node failure and degradation, mobility of sensor nodes, and changes in task and network requirements. They enable sensor nodes to be agile,

© 2005 by Chapman & Hall/CRC

self-aware, self-configurable, and autonomous. Nodes are aware of their own capabilities and those of other nodes around them that may provide the networking and system services or resources that they need. Although nodes are autonomous, they may cooperate with one another to disseminate information or assist each other in adapting to changes in the network configuration. An impromptu community of these nodes may cooperate to provide continual coordinated services, while some nodes may be newly deployed or removed from the spontaneous community.

Three fundamental mechanisms that support these self-organizing capabilities are service lookup, sensor node composition, and dynamic adaptation. Through a distributed implementation of these lookup servers, composition servers, and adaptation servers, other network and system services can be deployed and reconfigured spontaneously in the sensor network. They also dynamically adapt these services to device failure and degradation, movement of sensor nodes, and changes in task and network requirements. Application-specific network and systems services may be provided impromptu by sensor nodes and supporting nodes, including location services, naming and binding services, application-specific information dissemination and aggregation, caching and hoarding services, and security services. Critical sensor information can be disseminated through mobile transactions and dynamic query processing modules supported by the appropriate distributed services and network protocols to solve the problems of mobility, dispersion, weak and intermittent disconnection, dynamic reconfiguration, and limited power availability.

These distributed services enable sensor networks to overcome many of the following problems of very large and unstructured sensor networks that behave differently from traditional well-structured computer networks. First, many different types of sensor with a range of capabilities may be deployed with different specialized network protocols and application requirements. Data-centric network protocols are becoming common in sensor networks [4,5]. With many mixed types of sensor and applications, sensor networks may need to support several data-centric network protocols simultaneously.

Second, these mixed types of sensor node may be deployed incrementally and spontaneously with little or no preplanning. The networks must be extensible to new types of sensor node and services. They must be deployed spontaneously to form efficient *ad hoc* networks using sensors with limited computational, storage, and short-range wireless communication capabilities. They rapidly coordinate with each other to detect, track, and report activities and disseminate the information efficiently through the impromptu network of sensors.

Third, the sensor network must react rapidly to changes in the sensors composition, task or network requirements, device failure and degradation, and mobility of sensor nodes. Sensor devices may be deployed in very harsh environments, and be subject to destruction and dynamically changing conditions. The configuration of the network will frequently change due to constant changes in sensor position, reachability, power availability, and task requirements. The network protocols must be survivable in spite of device failure and frequent real-time changes. Sensor networks must be secure in the face of this open and dynamic environment.

Since many services are application specific, different protocols for a certain service may be specified for different applications. However, they may interoperate through the three fundamental mechanisms provided in the self-organizing sensor network architecture. For instance, in some sensor network applications, negotiation methods [6] may be preferred for information dissemination. Selected sensor nodes will register these negotiation services with the lookup server. Each of these sensor networks may establish their own services spontaneously and independently. Two different sensor networks may interoperate through filtering and translation services that must be defined to route information between the two sensor networks.

42.3 Preview of Existing Distributed Services

Distributed services are useful in environments where sensor devices must be quickly and flexibly deployed in large numbers to coordinate through impromptu networks. Each sensor device must

© 2005 by Chapman & Hall/CRC

operate autonomously in determining the capabilities of the sensor nodes in the vicinity and participate with the entire community of sensors to achieve global objectives. Distributed lookup services allow remote sensor nodes to be located more efficiently. The end-to-end and group communication services between nodes over a wide area are more efficient when localized routing algorithms are restricted within clusters. Latency could be lowered for wide-area communication involving very large numbers of sensors. Discovery of services in mobile systems provides critical support for self-organizing sensor systems when sensors are being deployed and removed on the fly. In Jini [7], service discovery relies on mobile Java codes and is implemented based on Transmission Control Protocol (TCP) and User Datagram protocol (UDP). It is not clear how these may be implemented using data-centric, *ad hoc* sensor networks with services based on more generic mobile codes. Service Location Protocol (SLP) [8] is an IETF protocol for service discovery that is designed solely for Internet Protocol (IP)-based networks. Bluetooth [9] devices have a range of 10 m and can directly communicate with at most seven other Bluetooth devices in a piconet. Bluetooth Service Discovery Protocol (SDP) [9] allows devices to browse and retrieve services by matching service classes or device attributes. Only services within the range of the device are returned. Our lookup service may retrieves services that could be multiple hops from the requesting node.

Existing services in reconfigurable middleware [10], such as adaptive CORBA [11,12], Jini [7] and XML [13], require much larger memory and power than are available in most micro sensor nodes. We instead develop lightweight distributed services that will execute with limited memory and computation power on micro sensor nodes. Furthermore, existing adaptive middleware assumes a fixed TCP/IP network and will not work in *ad hoc* sensor networks.

Many self-organizing sensor network services have focused on *ad hoc* network routing and localized algorithms to route information in an energy efficient way through autonomous sensor devices. The directed diffusion routing protocol [14], based on the localized computation model, provides energy-efficient and robust communication for dynamic network with small incremental changes. For dynamic networks with large-scale changes and a high level of mobility, directed diffusion may not adapt very well. A similar diffusion routing concept has also been presented in Hyphos [15]. Another localized protocol for information dissemination in sensor networks uses meta-data negotiation to eliminate redundant transmission [6]. Other self-organizing network routing protocols are dynamic source routing [16] and destination-sequenced distance vector [17], although it is not clear whether these algorithms are energy efficient enough for sensor networks. Our distributed services allow sensors to form high-level clusters and use directed diffusion within clusters. Clusters may be formed using localized algorithms [4] for coordinating among sensors to elect extremal sensors.

Distributed services may improve the efficiency of data-centric communication in sensor networks. Data-centric communication is useful, since the identity of the numerous sensors is not as important as the data they contain [4,5]. The networking infrastructure may provide more efficient dissemination of data through replication, caching, and discovery protocols [5]. Communication protocols must be energy efficient, since sensors [1,4] have very limited energy supply. In our architecture, sensors may cache and aggregate information. The discovery of these sensors can be made through the distributed lookup servers that are implemented using diffusion routing. Changes in the sensor network are propagated to other caching and aggregation services through the adaptation servers. Our framework facilitates consistent adaptation of networking and system services, as well as distributed sensor applications. Unlike other centralized control networks [18], our servers are associated only with sensors in a vicinity. Servers in different clusters will coordinate among themselves through information diffusion.

Recent advances on continual query and active databases can be exploited for remote surveillance in sensor networks through distributed services. The distributable interoperable object model (DIOM) system [19] is an object-based database designed primarily for integrated access to heterogeneous data sources. We extend this system to support sensor data sources and mobile nodes. DIOM continual queries may repeatedly retrieve and update sensor data for target tracking purposes. Cougar [20] is a distributed database designed specifically for networks of sensors. Sensor devices are ADT objects in an

© 2005 by Chapman & Hall/CRC

object-relational database. Sensor ADTs may contain asynchronous methods for retrieving readings from multiple sensors. Database operations, such as join, may be modified for these asynchronous methods.

42.4 Architecture of a Distributed Sensor System

We use an approach that integrates three mobility-aware system layers (Figure 42.1):

1. Application systems layer, e.g. sensor information-processing layer and collaborative signal processing.
2. Configurable distributed systems layer, which provide distributed services to the application systems.
3. Sensor networking and physical devices layer, which routes messages through the *ad hoc* sensor network.

The architecture avoids duplication of functionalities in the different layers and promotes efficient coordination between them. The sensor information layer contains collaborative signal-processing applications, mobility-aware mediators, and adaptive sensor query processing. The run-time reconfigurable distributed system contains distributed services for supporting mobile sensor applications. The network and physical layer contains data-centric network routing protocols, physical wireless transmission modules, and sensors that generate the raw data.

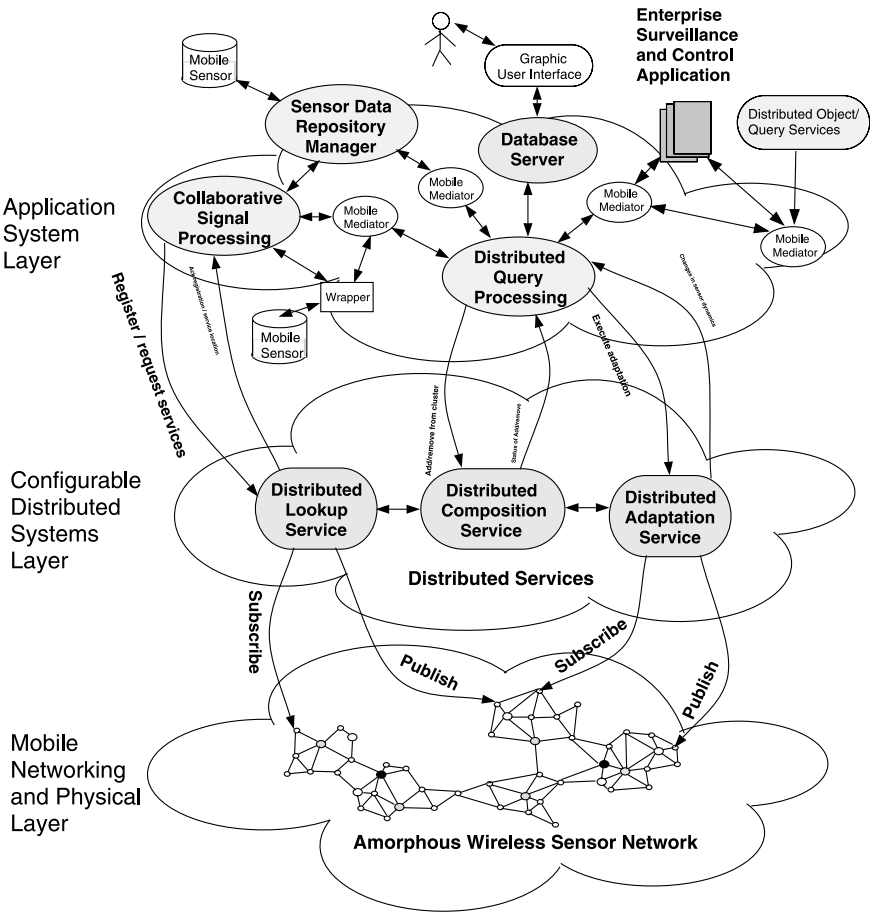


Figure 42.1. Architecture of a self-organizing distributed sensor system.

© 2005 by Chapman & Hall/CRC

At the physical device layer, different physical sensors and mobile devices may be assembled impromptu and reconfigured dynamically in an *ad hoc* wireless network. Each sensor node contains a battery power source, wireless communications, multiple sensing modality, a computation unit, and limited memory. Dual processors may be included for computation and real-time sensor processing. Three common sensing modality are supported: acoustic sensing using commercial microphones, seismic vibration using geophones, and motion detection using two-pixel infrared imagers. Wireless transceivers in the nodes provide communication between nodes, using time division multiplexing and frequency hopping spread-spectrum. Neighboring nodes in each cluster communicate through a master node that establishes the frequencies used by the nodes. Each node contains a global positioning system (GPS) receiver that allows the node to determine its current location and time. The GPS uses triangulation method with signals received from three satellites, to calculate the location of the node with an accuracy of 1 m. However, without clear line-of-sight to the satellites, as in urban environments, the GPS cannot be used. As we describe below, message routing and query processing use this location information.

At the networking layer, *ad hoc* routing protocols allow messages to be forwarded through multiple physical clusters of sensor nodes. Directed diffusion routing is used because of its ability to adapt dynamically to changes in sensor network topology and its energy-efficient localized algorithms. To retrieve sensor information, a node will set up an interest gradient through all the intermediate nodes to the data source. Upon detecting an interest for its data, the source node will transmit its data at the requested rate.

The configurable distributed system uses the diffusion network protocol to route its messages in spite of dynamic changes in the sensor network. These distributed services will support applications systems, such as distributed query processing, collaborative signal processing, and other applications. The advantage of using these services is that application and system programs may use simpler communication interfaces and abstraction than the raw network communication interface and metaphor (e.g. subscribe/publish used in diffusion routing). Furthermore, these distributed services may enhance the overall performance, such as throughput and delay. These services will be implemented on top of the directed diffusion protocol that can still be used by applications concurrently with these distributed services. Directed diffusion can still be the preferred method for retrieving sensor data and will be used by some of the services. On the other hand, distributed services provide other forms of communication — such as interpersonal communication and impromptu establishment of community of services — required by other applications.

At the application system layer, distributed query processing and collaborative signal-processing modules communicate with each other to support the surveillance and tracking functions of the enterprise. In sensor information systems, the cooperation between mobility-aware mediators, sensor agents, and collaborative signal-processing modules provides efficient access to diverse heterogeneous sensor data, surveillance, and tracking information through the sensor network. The mobile sensor information layer is supported by three major components: interoperable mobile object, dynamic query processing, and mobile transactions. We will not discuss mobile transactions in this chapter. In the interoperable mobile object model, the cooperative network of mobility-aware mediators and sensor agents will be configured to support interfaces to remote sensor data sources through multi-hop wireless network protocols.

42.5 Data-Centric Network Protocols

We use the directed diffusion protocol [14] to implement all the distributed services and for retrieval of data through dynamically changing *ad hoc* sensor networks. Diffusion routing converges quickly to network topological changes, conserves mobile sensor energy, and reduces the network bandwidth overhead, since routing information is not periodically advertised. Routing is based on the data contained in sensor nodes rather than unique identification. Directed diffusion is a type of reactive routing protocol which only updates routing information on demand. In contrast, proactive routing

© 2005 by Chapman & Hall/CRC

protocols, such as links state routing, frequently exchange routing information. For sensor networks that experience greater dynamic changes, reactive routing algorithms are more appropriate, whereas proactive routing algorithms are more efficient for those that are more static and experience infrequent topological change.

Directed diffusion is a data-centric protocol, i.e. nodes are not addressed by IP addresses but by the data they generate. A node names the data it generates by their *attribute–value* pairs. A sink node requests for a certain data by broadcasting an *interest* for the named data in the sensor network. The interest and gradient is established at intermediate nodes for this request throughout the sensor network. When a source node has a data that matches the interest, the data will be "drawn" down towards that sink node using this interest gradient that was established. Intermediate nodes may cache, transform data, or direct interests based on previously cached data. The sink node can determine if a neighbor node is in the shortest path whenever it receives new data earliest from that node. The sink node will reinforce this shortest path by sending a reinforcement packet with a higher data rate to this neighbor node, which then forwards it to all the nodes in the shortest path. Other nonoptimal paths may be negatively reinforced, so that they do not forward data at all or do so at a lower rate.

Distributed services and applications use publish and subscribe APIs provided by directed diffusion. Through the subscribe() function, an application declares an interest that consists of a list of attribute–value pairs. The subscription is then diffused through the sensor network. A source node may indicate the type of data it offers through the publish() function. It then sends the actual data through the handle returned from the publish() function. The sink node then receives the data that has propagated through the sensor network using a recv() function call with the handle returned from the subscribe() call.

42.6 Distributed Services for Self-Organizing Sensor Applications

By augmenting sensor nodes as reconfigurable smart nodes through distributed services, we can simplify the development of self-organizing networks. These smart sensor nodes may be developed independently but may interact with other smart sensor nodes. Some smart sensor nodes may execute autonomously to provide networking and system services or control various information retrieval and dissemination in the dynamically changing sensor network [21]. To enhance the ability to reconfigure their networking, configuration, and adaptation functionalities, smart sensor nodes may make use of three main classes of distributed services: lookup service, composition service, and adaptation service (Figure 42.1).

The lookup service enables new system and network services to be registered and made available to other sensor nodes. Methods for calling the services remotely are also provided. The composition service allows clusters of sensor nodes to be formed and managed. The adaptation service allows sensors nodes and clusters to reconfigure dynamically as a result of sensor node mobility, failure, and spontaneous deployment. These servers enable sensor nodes to form spontaneous communities in *ad hoc* sensor networks that may be dynamically reconfigured and hierarchically composed to adapt to real-time information changes and events.

These distributed servers may be replicated for higher availability, efficiency, and robustness. Distributed servers coordinate with each other to perform decentralized services, e.g. distributed lookup servers may work together to discover the location of a particular remote service requested by a node.

42.6.1 Reconfigurable Smart Nodes

By exploiting these distributed services, sensor nodes can be enabled to be self-aware, self-reconfigurable, and autonomous. These sensor nodes, known as reconfigurable smart nodes, can be used to build scalable and self-organizing sensor networks. (In this chapter, we refer to reconfigurable

© 2005 by Chapman & Hall/CRC

smart sensor nodes as smart nodes or sensor nodes.) Smart nodes may represent sensor nodes, other types of mobile node, fixed nodes, or clusters of these nodes. They may simultaneously be service providers for other smart nodes and clients of services that other smart nodes provide. Smart nodes may be dynamically composed into impromptu networked clusters forming clustered smart nodes that work together to provide abstract services for the agile sensor network. They may also adapt rapidly to abrupt changes in the sensors, capabilities, events, and new real-time information. Very large networks with hundreds of thousands of sensors nodes can built by hierarchically composing reconfigurable smart nodes.

Smart sensor nodes may consist of hardware devices and software for interacting with the real-world systems. The hardware may contain computational, memory, wireless communication, and sensing devices. Smart nodes may contain control software for monitoring information from real-world devices such as simple sensors, engaging in distributed signal processing and generating appropriate control signals to produce a desired result in the real-world system. The control software takes advantage of the functionalities provided by the networking and system software.

Smart nodes interact with other smart nodes through well-defined interfaces (for networking and systems operations) that also maintain interaction states to allow nodes to be reconfigured dynamically. These explicit interaction states and behavior information allow localized algorithms with the adaptation servers to maintain consistency when autonomous nodes and clusters are reconfigured dynamically, move around, or recover from failure. Smart node implementation and data (software and hardware) are encapsulated (hidden) from other nodes.

Different designers may independently develop smart nodes and their network and system services using different methods. For example, one designer may use a network protocol that is suited for a particular sensor application with its set of network requirements, such as low latency, power conservation, GPS capability, high error rate, and disconnection. In order to ensure consistency during dynamic reconfiguration and failure recovery of sensor nodes, adaptation servers may analyze the protocols using their specification model.

When new smart nodes are added to the sensor network, they register their services with a lookup server (Figure 42.2). Other nodes that require a service will discover the services available in a cluster through the lookup servers that return the location and interface of the service nodes. This is similar to

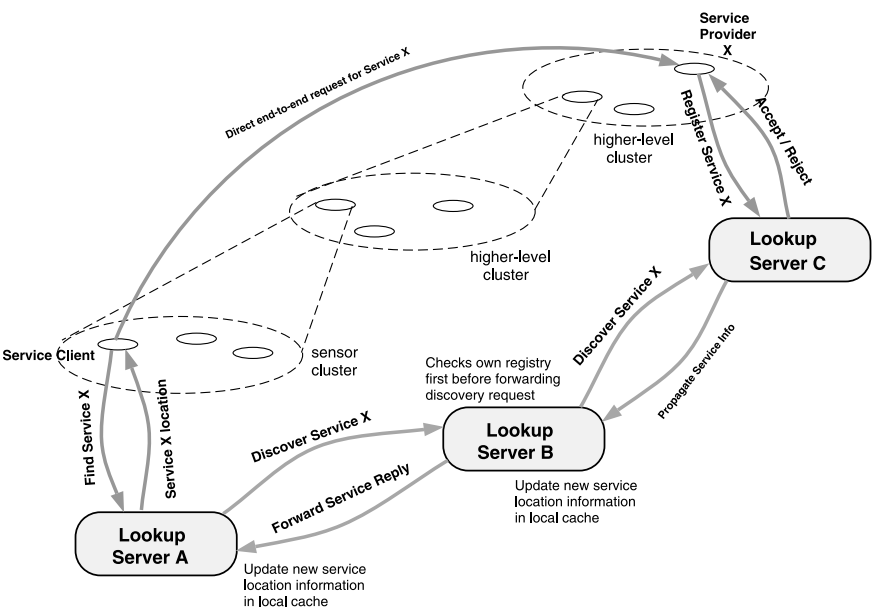


Figure 42.2. Discovery of services with distributed lookup server.

© 2005 by Chapman & Hall/CRC

Jini [7], which manages system-level services based on Java code executing in IP-based networks. On the other hand, reconfigurable smart nodes may provide lower level networking services using generic mobile codes executing in data-centric sensor networks. Client nodes then interact directly with the service node. Smart nodes are self-aware of their own location, configuration, and services that they perform.

42.6.2 Lookup Services

A sensor node may deploy new network and system services for use by other nodes in a self-organizing sensor network. A sensor node that provides, a service is called a service provider, and a node that uses the service is called a service client. Since service providers may be introduced or removed from the sensor network at any time, a lookup server is needed to keep track of the availability of these services. A sensor node may register a resource that it maintains or a service that it can perform with a lookup server (Figure 42.2). Each smart node has a home lookup server that keeps track of the location of the node when it moves. A lookup server may contain information on services or resources at multiple clusters. Other nodes that require the service may request the service through a lookup server. If the service is recorded in the lookup server, then it will return the location of that service to the requesting node. Otherwise, if the service is not recorded in the lookup server of the region, then a discovery protocol is used to locate the service through other lookup servers (Figure 42.2). A request message is propagated to all the lookup servers and the server that contains the service registration information will return the reply with the service location. It may also return the cluster name of that service. The lookup server that made the request will then cache that service location and cluster name information in its local registration cache. At regular frequency, service and resource registration information may be disseminated from one lookup server to other lookup servers in the agile sensor network.

Lookup servers support mobility of sensor nodes. When a sensor node moves to a different cluster at another location, whenever possible it notifies the previous lookup server that it is moving. When it arrives at another cluster in a new location, it will register with the new lookup server, which will notify the previous lookup server (Figure 42.3). The new lookup server will propagate the change in the nodes' location to other lookup servers. The lookup server responsible for a sensor node that is interacting with

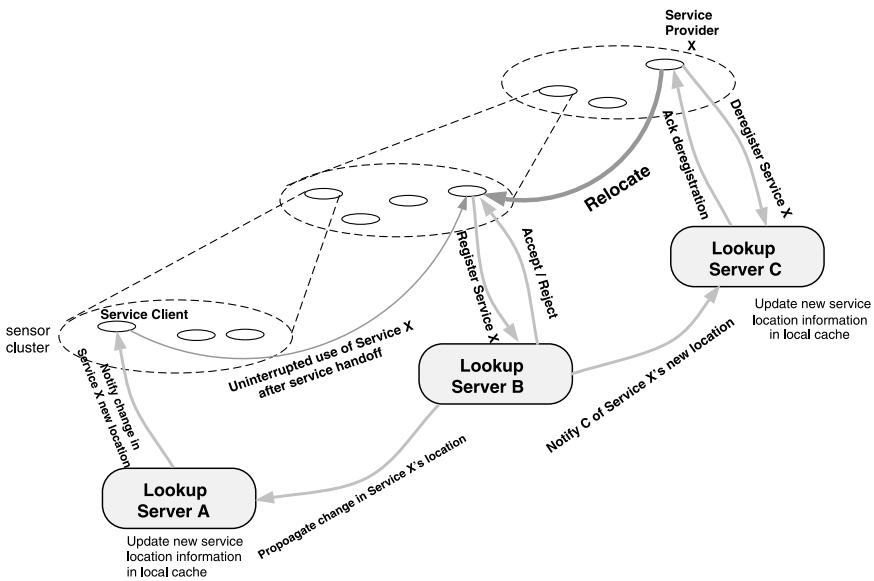


Figure 42.3. Mobility of smart nodes.

© 2005 by Chapman & Hall/CRC

the mobile node will notify the sensor node of the service location change. Existing interactions between the mobile node and other nodes will thus be handed over to the new location. Adaptation servers may be involved in the handover operation to preserve global consistency during the handoff, as discussed below, resulting in uninterrupted use of the service.

42.6.3 Composition Services

The compositional server manages various smart nodes that may be added to (or removed from) clusters in the agile sensor network. It also manages network abstractions (or group behavior) of clusters and hierarchical composition of clusters. The compositional server simplifies dynamic reconfiguration of services provided by each smart node or cluster. It also simplifies the development of a large self-organizing sensor network by allowing individual nodes and clusters to be specified and designed independently while the compositional behavior and constraints on a cluster of components may be specified separately.

Compositional servers enable compositionality and clustering abstraction of sensor networks. To enhance adaptivity in sensor networks, each node is designed independently and the networking requirements with other nodes may be specified separately. This decoupling of autonomous smart nodes from their networking requirements enables smart nodes to be easily adapted, replaced, and reconfigured when triggered by dynamic events in the sensor network.

Clusters of smart sensor nodes may be formed under the management of a compositional server. Hierarchical clusters are also possible for larger scale sensor networks. A cluster of sensors may also provide distributed services by coordinating the tasks among the sensors, such as aggregating summary information. Clustered smart nodes encapsulate the networking and system capabilities provided cooperatively by the group of smart nodes. There will be a head smart node in the cluster that is responsible for the control of the cluster and inter-cluster communications and networking functions. Group communication to nodes in a cluster can be efficiently implemented by sending a message first to the cluster head, which then multicasts it to the member nodes. Member nodes will elect a cluster head from the set of nodes with most powerful networking and system capabilities. Smart nodes in a cluster may cooperate to perform the networking and system functions for the cluster. Synchronization constraints associated with network protocols and system services among smart nodes may be specified in clustered smart nodes. The capability to specify hierarchical composite clusters enables designers to build large and complex sensor networks by clustering together smaller network-enabled sensor devices at each level.

42.6.4 Adaptation Services

Adaptation servers utilize information from the compositional server, lookup server, and analytical tools to control smart nodes during dynamic reconfiguration and failure recovery. Each smart node may execute autonomously to control different network operations in the sensor network and may interact and coordinate independently with other smart nodes to perform collaborative networking operations.

Adaptation servers monitor clusters of smart nodes during normal execution either through the spontaneous signal from the sensors, probing of the smart nodes, or explicit network management directives for reconfiguration and failure recovery. When a runtime reconfiguration is requested or triggered, the adaptation server will generate the appropriate schedule of reconfiguration operations that will ensure the reconfigured and affected sensor nodes are globally consistent. To ensure correct adaptation and maintain consistency, the adaptation server makes use of analytical tools for dependency analysis and relevant information from compositional servers and lookup servers. When smart nodes are added or removed from the agile sensor network, a suite of analytical tools may be utilized to ensure that the sensor network still maintains its safety and liveness properties [22]. Smart nodes (or clusters of smart nodes) may be specified and analyzed independently.

© 2005 by Chapman & Hall/CRC

42.6.5 API for Lookup Services

Applications use the lookup service through the following API. We focus primarily on the lookup service API in this chapter, since it is the main service that is responsible for enabling sensor networks to be self-organizing. In the next section we will describe how these API functions are used by the various application systems of a surveillance sensor network.

These API functions use the following parameters.

- service_type is the generic type of service for which there may be several instances. For example, a type of service may be temperature monitoring, whereas specific instances of temperature sensors may be sensor *X* at location *Y*.
- service_name is the specific name that identifies an instance of a service provider.
- input_list is a list of attribute–value pairs containing the input parameters to the service invocation.
- output_list is a list of attribute–value pairs containing the output values from the service invocation.
- lifetime is the time period in which service information will be stored in a lookup server.
- interface_type is one of the following three types of interface that the callee of the service use:

  1. Location or address. This is used by the service client if the interface for interacting with the service provider is known. The service client only needs to retrieve the location or address of the service provider to be used for invoking the service request.
  2. Interface definition. This is used by the service client to retrieve the definition of the interface for interacting with the service provider. The service client must have the interpreter or compiler for the interface definition.
  3. Mobile code. The service client retrieves the mobile code that implements the protocol for interacting with the service provider. The mobile code must then be dynamically linked to the service client.

Nodes that receive the service provider information can make service calls through a method provided below. The following is a description of the purposes and the side-effects of the lookup service function calls.

1. service_call(service_name, input_list, output_list, interface_type)

   This function allows a service client to find and make a call for a service when the service client does not know the location or address of the service provider and/or the interface for using the service. It is implemented as a combination of the lookup_service() and service_exec() calls described below. This function requires a specific service name to be provided. A generic service type cannot be used, since several service provider instances may match the service type.

2. status = lookup_service(service_type, service_name, input_list, output_list, interface_type)

   This function allows a service client to find the location or address of a service provider and/or the interface for using the service. If service_type is defined and service_name is NULL, then all service providers registered with the lookup server of that type are returned. Service lookup can also be based on cluster or predicate matching. The cluster information and predicate for matching service providers are contained in the input_list. Depending on the interface_type used, i.e. location, interface definition, or mobile code, the respective results of the lookup_service() call will be placed in the output_list.

3. status = service_exec(service_name, input_list, output_list, interface_type)

   This function allows a node to request for the service and gets the results back from the service provider. The input parameters are specified by the client in the input_list. The service provider

© 2005 by Chapman & Hall/CRC

performs the requested service or remote procedure call and returns the results in the output_list. The interface_type defines the method used by the service client to communicate with the service provider.

4. status = service_register(service_type, service_name, lifetime, input_list)

   This function allows a service provider to register its service with a lookup server in the region. Services will remain in the lookup server for the lifetime specified by the service provider. In the input_list, the service provider may supply one or more of the following service information: (i) location or address, (ii) interface definition, or (iii) mobile code. Lookup servers for different regions may coordinate with each other to update their lists of service information.

5. status = service_deregister(service_type, service_name)

   This function allows a service provider to remove its service from the lookup server registry.

42.7 Application Systems

The self-organizing sensor network architecture allows sensor information system designers to specify their own specialized protocols and services that are most appropriate and efficient for the specific application, although most sensor nodes may use generic protocols and services. Since these services may be application specific, different protocols for a certain type of service may be specified for different applications. Each of these sensor applications may establish their own services spontaneously and independently of each other. Various types of network service may be independently defined for different sensor nodes. In the following subsections, we discuss some of these types of services.

42.7.1 Interoperable Mobile Object Model

The interoperable mobile object model is useful for retrieval of sensor information from sensor data sources to the surveillance application clients. The interoperable mobile object model extends the DIOM [19] to sensor information networks. Application clients may access and update information sources in the sensor nodes through a group of mediators, object servers, sensor data repository managers, and sensor agents which communicate with each other using distributed services and diffusion routing (Figure 42.4).

An application end-user poses queries to the database server which coordinates with the mediators to decompose, schedule, and route queries to the sensor information sources and collaborative signal-processing modules. Mediators resolve the bindings of the sensor data sources through the object servers. The binding maps the object unique identifier to sensor agents or a repository manager at the sensor nodes specified by their location, data types, and application-specific information. Once the binding is resolved, mediators communicate with the sensor nodes directly. When sensor nodes move to new locations, routes may be changed through the directed diffusion protocol if the change is incremental or through the distributed lookup service if the node moves quickly over a longer distance. Directed diffusion and the lookup services are responsible for solving the problem of mobility at the network routing and distributed service levels. However, applications services, such as mediators and object servers, must also be aware of the mobility of sensor data sources and cache location binding information. Distributed lookup servers notify mediators and object servers of relocation of sensor nodes.

42.7.1.1 Mediators

Mediators cooperate with each other to implement dynamic query processing and mobile transactions in sensor networks. When a database server first receives a query, it may decompose it into multiple subqueries and forward them to selected mediators that may be associated with sensor object agents or collaborative signal-processing agents for answering each subquery. Sensor agents (described in

© 2005 by Chapman & Hall/CRC

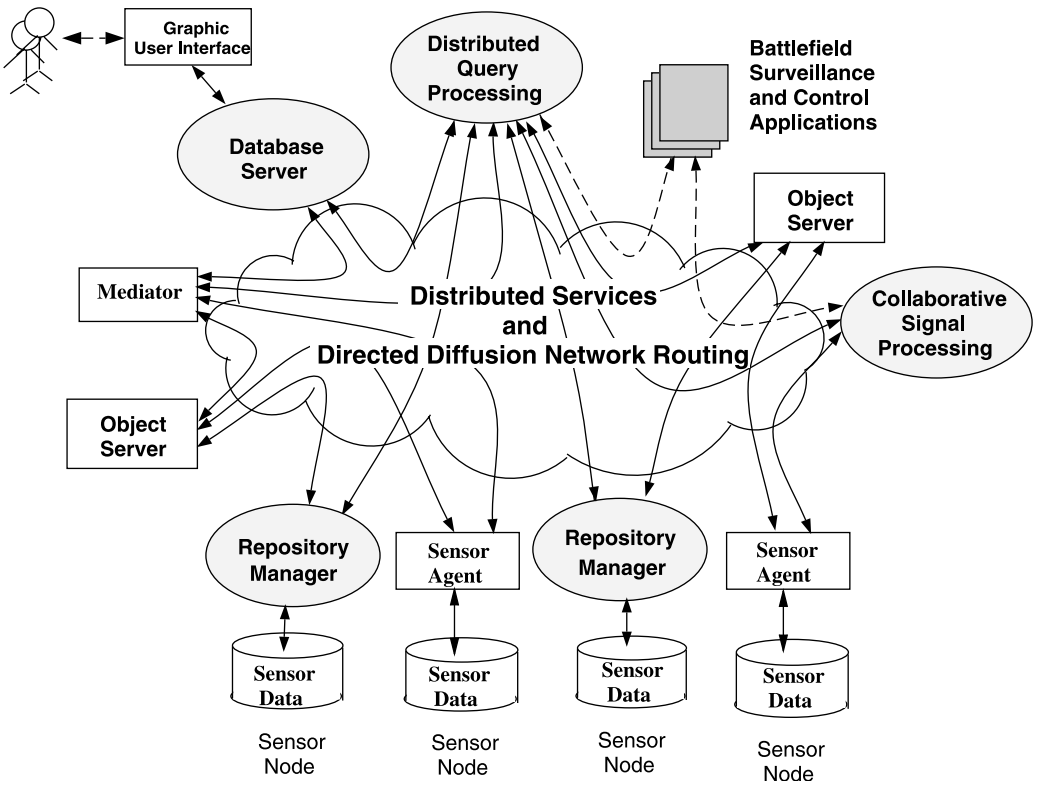


Figure 42.4. Distributed services support for distributed query processing and collaborative signal processing.

Section 42.7.1.3) provide accesses to their local sensor data sources. Collaborative signal-processing agents (described in Section 42.7.2) provides more accurate tracking and surveillance data by combining readings from multiple sensors. Each mediator may dynamically determine and update its association with the sensor agents and signal-processing agents in the region. A mediator may discover the sensor agents and signal-processing agents in its region by calling lookup_service() using a generic service_type() parameter to retrieve all agents that provide the type of service required to process the subquery.

The selection of the mediators for each subquery is based on the database server's knowledge, stored in its cache, of the current location of the sensor data sources involved in the query. If the database server does not have location of the mediators, then the database server may locate the selected mediators using the lookup_service() call. It then calls service_exec() to execute the subqueries in each selected mediator. Results of the subqueries are returned from these service_exec() calls. The database server may alternatively use service_call(), which may automatically perform all the operations above by locating, executing the subqueries, and retrieving the results from each of the selected mediators. Mediators may be grouped into clusters using the composition service. Clusters may represent logical data partition and enable more efficient group communication to a cluster of mediators.

Dynamic query processing is performed within each mobile transaction. While the initial mediator is responsible for the overall transaction, the subsequent mediators are responsible for the subtransactions and their related locking and logging functions. Mediators are aware of mobility of sensor nodes through notification from the distributed lookup service. Sensor nodes that have moved to a new location may register, using register_service(), with another lookup server. The new lookup server will update the previous lookup server by sending a notification event to the mediator that is accessing

© 2005 by Chapman & Hall/CRC

the sensor node information. When sensor nodes have moved, mediators must reconfigure the routing and scheduling of their subqueries to current locations of the sensor nodes. In addition to mobility, mediators may reconfigure their subqueries when they detect disconnection and bandwidth variability.

42.7.1.2 Object Server

Object servers are primarily responsible for caching information on the location, availability, and attributes of sensor data sources. When a mediator is interested in retrieving particular sensor information, it can either inject an interest through the diffusion protocol or try to locate the sensor data source from an object server. The first option is preferred if the target sensors are within a certain range. The second option is preferred if the target sensors are located farther away, in which case the object server will return the location and application-specific attributes of the sensor data sources. Once the sensor data sources are located, the mediator communicates with them directly using directed diffusion.

Each mobile sensor node has a *home object server*. A large sensor network may contain many object servers, each responsible for a subset of sensor nodes. Each object server registers its service with a lookup server in its region using the service_register() call. This enables other sensor nodes to locate and register their object information with the object server. The home object server caches the current locations of sensor nodes whenever they move. If the sensor node moves to a new location by a short distance, the directed diffusion protocol will handle mobility and change in the network topology. Location updates at the home object server are unnecessary for mobility over a short distance. However, for mobility over a longer distance, sensors that have moved will try to update their location with their home object server. They first searches for the object server using the lookup_service() call. Several lookup servers may be used to locate the object server. Then they update their location with the object server using the service_exec() call. Object servers keep track of location and application-specific information of objects, whereas lookup servers keep track of the location and system-level information of services.

In continual queries, data are continually sent from the sensor nodes to the mediators through the service_exec() call. When sensor nodes move over a large distance, they will update their locations with their home object servers through the distributed lookup services. They also update their information with the lookup server in the current region, which will propagate the information of all distributed lookup servers. Changes in the sensor location will trigger the handover procedure in the service_exec() call, which will redirect the distributed service call, to retrieve data from the new sensor location. The update of the location cache in the object server will improve the performance of future queries.

If a sensor node's movement affects the performance of an on-going continual query, then the mediator may re-evaluate the query decomposition. Alternate query routes and schedules may be used for updates and queries to sensor nodes that have moved by tunneling subqueries directly to the foreign mediator. Early detection and notification of mobile sensor relocation by the lookup servers allows query routes and schedules to be updated to reflect current sensor node locations, thus improving the performance of distributed queries.

42.7.1.3 Sensor Agents

A sensor agent is a software module that serves one sensor data source. The sensor agent is built around the existing sensor data source to turn it into a local agent for the sensor object in order to make an existing sensor information source available to the network of mediators. Sensor agents enable different types of sensor to be deployed incrementally in the *ad hoc* sensor network. The local agent is responsible for accessing the sensor information source and obtaining the required data for answering the query. Sensor agents are customized to integrate with applications methods used in the sensor nodes, such as autonomous identification and location management. Services provided by sensor agents may also include translating a subquery into a sensor information retrieval command or language expression,

© 2005 by Chapman & Hall/CRC

submitting the translated query to the target sensor information source, and packaging the subquery result into a mediator object.

The sensor agent is also responsible for local management for sensor data stored in the node. It contains the local locking mechanisms for the global concurrency control scheme and the local recovery mechanism based on write-ahead logs. Agents may convert data in raw format to interoperable sensor object format, turning each sensor into an agent for the interoperable mobile object.

Each sensor agent registers with a home object server and supplies all its related application-specific object information, such as object identification and location. Other processes, such as mediators, may then locate the sensor objects through the object server. A sensor agent also register its service with the lookup server through the service_register() call in order to handle sensor mobility and reconfiguration more efficiently at the distributed services level.

When the sensor agent moves, it will inform the object server of its current location. However, it must first determine how to reach the object server through the lookup_service() call and then update the home object server through the service_exec() call. The object servers may, in turn, notify the mediators of the change in sensor movement or connection variations. The mediators may determine whether to modify the subquery schedule.

42.7.2 Collaborative Signal Processing

Retrieval of sensor values from multiple sensors can increase the accuracy of the data in target recognition and tracking [23]. The multiple reading of the sensor values can be statistically combined to derive more accurate tracking data. Sensors in a region may be clustered together through the distributed composition server, where their sensor reading may be combined using a weighted voting algorithm [24] to provide more accurate data. For each cluster, a sensor node may be elected to be the head of the cluster. Readings from multiple sensors in the clusters will be propagated to the cluster head through diffusion routing where the algorithm is applied. Results from each cluster head may be propagated to higher level cluster heads for data fusion.

In many sensor applications, the data generated by the sensor node may be very large. The cost of propagation of these sensor data, as in the previous scheme, will be prohibitive. One solution to address this problem is to use mobile agents to migrate from node to node to perform data fusion using the local data [25]. Instead of transferring large amounts of data throughout the sensor network, this approach only transfers the mobile-agent code, which is smaller than the sensor data. The result is an improvement in the execution time of the collaborative signal-processing algorithm.

A sensor node that needs to execute a mobile-agent code must download the mobile-code from a mobile code repository manager in its region. It may use the lookup_service() call to first locate the repository manager that stores the relevant mobile code. It then calls service_exec() to the repository manager to download the code. An alternative method is to store the mobile code with the lookup server.

The sensor node can then use lookup_service() to retrieve the mobile code directly from the lookup server.

The entire surveillance area is split into several subareas. Each subarea is controlled by a signal-processing agent, also known as processing element (PE), which may dispatch several mobile agents into that subarea. The signal-processing agent may register itself with the lookup server using service_register() to allow other nodes, such as the mediators, to access its tracking results. Each mobile agent will migrate from node to node to perform data fusion, for instance using a multi-resolution data integration algorithm [25]. Each mobile agent, addressed by an identification number, contains an itinerary, data, method, and an interface. The identification is a 2-tuple composed of the identification of the dispatcher and the serial number assigned by the dispatcher. The itinerary describes the migration route assigned by the dispatcher.

The data is the agent's private data that contains the integration results. The method describes the multi-resolution data integration algorithm. The interface is the function by which the agent communicates with the PE and for the PE to access the agent's private data.

© 2005 by Chapman & Hall/CRC

When mobile agents migrate from node to node, the results of the sensor integration algorithm from previous nodes are cached in the mobile agents. The state information must be transferred with the mobile agent as it migrates from node to node. Agent migration and state transfer are supported by the distributed adaptation service. As the mobile agents visit each node, they may register with the lookup server. The primary signal-processing agent (PE) may find these mobile agents as they move around and retrieve results using the service_call() function. A mobile agent may also retrieve intermediate results from other mobile agents through service_call().

For migration of mobile agents far beyond a region managed by a lookup server, the agent will register with another lookup server in the new region. A service client may try to contact the mobile agent through the first lookup server but may need to use several intermediate lookup servers to get the information about the mobile agent current location.

Multi-resolution signal-processing algorithms may be implemented using hierarchical clustering of sensors provided by the distributed composition server. Results from a sensor cluster may be passed to agents responsible for signal processing for higher level clusters. Agents for higher level clusters may send messages to multiple sensor clusters using the group communication method provided by the service_exec() call by specifying the appropriate service_type parameter.

42.8 Conclusions

We have described how distributed sensor applications, such as sensor information retrieval and remote surveillance, can be supported by distributed services in self-organizing sensor networks. The three basic distributed servers are lookup servers, composition servers, and adaptation servers. Through these servers, sensor nodes may be placed together impromptu in spontaneous environments and these sensor nodes will immediately know about the capabilities and functions of other sensor nodes and work together as a community system to perform cooperative tasks and networking functionalities. Newly deployed sensor nodes may provide new services. Other sensor nodes may locate and use these services spontaneously. These distributed services are implemented using directed diffusion network routing, which provides energy-efficient and data-centric data communication. While diffusion routing can adapt dynamically to limited mobility and topological change, the distributed services supports large mobility and changes in sensor nodes. Diffusion uses a data-centric communication model, whereas it may be more convenient to use end-to-end process-oriented communication in distributed sensor applications. Sensor nodes that discover services provided by other sensor nodes may call these services either well-known interfaces, interpreted interface definition, or mobile codes downloaded from the lookup server. The benefits of using these distributed services are that application and system programs may use simpler communication interfaces and abstraction rather than the raw network communication interface and metaphor of the sensor network layer (e.g. subscribe/publish used in diffusion routing). Furthermore, these distributed services may improve the overall performance, such as throughput and delay.

Acknowledgments

This research is supported in part by the Space and Naval Warfare Systems Center (SPAWAR), San Diego, California, and the DARPA SensIT program.

References

[1] Kahn, J.M. *et al.*, Next century challenges: mobile networking for smart dust, in *ACM Mobicom*, 1999.

[2] Pottie, G.J. and Kaiser, W.J., Wireless integrated network sensors, *Communications of the ACM*, 43(5), 51–58, 2000.

© 2005 by Chapman & Hall/CRC

[3] The Ultra Low Power Wireless Sensors project, http://www-mtl.mit.edu/~jimg/project_top.html, September 2004.
[4] Estrin, D. *et al.*, Next century challenges: scalable coordination in sensor networks, in *ACM Mobicom*, 1999.
[5] Esler, M. *et al.*, Next century challenges: data-centric networking for invisible computing, in *ACM Mobicom*, 1999.
[6] Kulik, J. *et al.*, Negotiation-based protocols for disseminating information in wireless sensor networks, in *ACM Mobicom*, 1999.
[7] Arnold, K. *et al.*, *The Jini Specification*, Addison Wesley, 1999.
[8] Guttman, E., Service Location Protocol: automatic discovery of IP network services, *IEEE Internet Computing*, 3(4), 71, 1999.
[9] Specification of the Bluetooth System; http://www.bluetooth.com/developer/specification/specification.asp.
[10] Agha, G., Adaptive middleware, *Communications of the ACM*, 45(6), 31, 2002.
[11] Schmidt, D.C., Middleware for real-time and embedded system, *Communications of the ACM*, 45(6), 43–48, 2002.
[12] Kon, F. *et al.*, The case for reflective middleware, *Communications of the ACM*, 45(6), 38, 2002.
[13] http://www.w3.org/XML, September 2004.
[14] Intanagonwiwat, C. *et al.*, Directed diffusion: a scalable and robust communication paradigm for sensor networks, in *ACM Mobicom*, 2000.
[15] Poor, R., Hyphos: a self-organizing, wireless network, Master's Thesis, MIT Media Lab., June 1997.
[16] Johnson, D.B. and Maltz, D.A., Dynamic source routing in *ad hoc* wireless networks, in *Mobile Computing*, Imielinski, T. and Korth, H. (eds), Kluwer Academic Publishers, 1996, 153.
[17] Perkins, C.E. and Bhagwat, P., Routing over multi-hop wireless network of mobile computers, in *Mobile Computing*, Imielinski, T. and Korth, H. (eds), Kluwer Academic Publishers, 1996, 183.
[18] Echelon, The LonWorks Company, LonWorks Solutions, http://www.echelon.com/Solutions/, September 2004.
[19] Liu, L. and Pu, C., The distributed interoperable object model and its application to large-scale interoperable database systems, in *ACM CIKM'95*, 1995.
[20] Bonnet, P. *et al.*, Querying processing in a device database system, Technical Report Tr99-1775, Computer Science, Cornell University, October 1999.
[21] Lim, A., Architecture for autonomous decentralized control of large adaptive enterprises, in *DARPA–JFACC Symposium on Advances in Enterprise Control*, San Diego, CA, November 1999.
[22] Lim, A., Automatic analytical tools for reliability and dynamic adaptation of complex distributed systems, in *Procedings of IEEE ICECCS*, Florida, November 6–10, 1995.
[23] Brooks, R.R., Modern sensor networks, in *CRC Handbook of Sensor Fusion*, CRC Press, 2002, chapter 26.
[24] Saari, D.G., Geometry of voting: a unifying perspective, in *Proceedings of Workshop on Foundations of Information/Decision Fusion with Applications to Engineering Problems*, DOE/ONR/NSF, Washington DC, August 1996.
[25] Qi, H. *et al.*, Distributed multi-resolution data integration using mobile agents, in *Proceedings of IEEE Aerospace Conference*, 2001.

© 2005 by Chapman & Hall/CRC

43

Adaptive Active Querying

Bhaskar Krishnamachari

43.1 Introduction

In the most abstract sense, a sensor network is a collection of nodes at which data are being produced (through sensing of the physical phenomena). For the network to be of practical use, at least some of this data or a processed version of it must be provided to at least one end user. The implementation of this information flow depends upon the application requirements, as well as the resources available. In the simplest kinds of sensor network, all the data generated may be sent continuously to a central data sink. This would naturally be resource intensive in terms of bandwidth and energy, and may not be necessary if the end application requires only processed notification of events. Therefore, more sophisticated systems opt to route only selective information based on the in-network processing of one or more nodes' data, upon the issuance of a request (query) for this information.

In traditional networks, the primary task of the routing layer is to provide end-to-end connectivity so that arbitrary applications on two data terminals in the network may communicate with each other. Wireless sensor networks are different from traditional open networks in two crucial respects: the first is that they are often severely energy constrained, and there is a great need for optimizing protocols to minimize communication costs (which are often several orders of magnitude more expensive than computation); the second is that sensor networks tend to be significantly more application specific, so that there is a much greater scope for optimizing protocols for a given application. These features of wireless sensor networks argue for the implementation of cross-layer mechanisms in this domain that are very different from traditional networks. The end-to-end address-centric abstraction in which node IDs are the primary attributes for information flow can be replaced by data-centric techniques that allow data attributes and application information in a packet to be taken into account while deciding where to send it next, rather than relying on network addressing alone.

In these data-centric techniques, there are three general patterns of information flow, two of which are significant. The first is a well-formulated query for data emanating from a data consumer node (referred to as a sink). The second significant information flow pattern is the response information from one or more data-producing nodes (sources) potentially aggregated en route. The third (not always used) is an advertisement of available data from the sinks, a sort of reverse query looking to see if there are nodes interested in this data. When the second information flow pattern dominates, in terms

© 2005 by Chapman & Hall/CRC

of the amount of information being sent, the details of how the first (and possibly the third) pattern are implemented do not seriously affect network performance. This occurs primarily when the queries are for a continuous flow of information from the sources to the sink. In such a case, the query could be simply flooded through the network, with the cost of the flooded query being amortized over the much larger cost of the information flow it initiates. However, when the queries are for noncontinuous, instantaneous, information, such a strategy could be very inefficient indeed. Such queries need to be handled more intelligently, or rather they could be *made* intelligent. This is the basic premise behind the idea of *active querying* that we will describe in this chapter — query packets that move through the network, actively searching for the information, resolving themselves partially as they pick up information, and deciding, based on their contents and the state of the nodes they pass through, where they should head next and when they should terminate.

From a user perspective, it makes sense to treat the sensor network as essentially a distributed database (e.g. see [1,2], and the pertinent sections of this book). The user submits potentially complex queries for information from the network through a standardized query interface. The query is then disseminated and a response obtained, possibly after in-network aggregation/filtering, such as by using operators (selects/joins). The distributed database perspective provides a unified way to handle different types of query and, most importantly, allows for flexibility in how the information flow and routing patterns for different queries and their corresponding responses are implemented. Active queries fit into this vision because they will constitute a component of a portfolio of querying and routing techniques used to implement in-network processing of queries.

To clarify the design space for which active queries are well suited, consider the following examples from different applications ranging from target tracking to habitat monitoring:

- Where is target X currently located?
- Has there been rainfall in any part of the region in the past hour?
- Where is the chemical concentration the highest?
- Give k locations where species X's calls have been recorded.
- Is the Boolean expression $(W \,\&\, X \,\&\, \neg Y) + (Z \,\&\, \neg X)$ true (where W, X, Y, Z are binary conditions/subqueries such as temperature > 50, humidity < 0.5, etc.)?

What is common to all these example queries is that they require a one-shot response based on current (or stored) data at the nodes. They may be complex (involving several subqueries), and for replicated data within the network (there may be several nodes in the network that can provide a response to the same query).

This chapter will focus on a discussion of different mechanisms that can be used to implement active querying in sensor networks. We begin with the simple idea of a random walk and describe how it is used for different active querying techniques. We then discuss the possibility of sending active queries on predetermined trajectories, the improvement of active query performance using reinforcement learning, and the use of geographic and sensor information to direct the query.

43.2 Active Queries as Random Walks

An active query, as we define it, refers to a packet that contains the query and which proceeds through the network moving from node to node in order find a response to the query. The simplest implementation of such a query is the execution of a random walk, in which the active query packet is forwarded to a random neighboring node at each step. This simple random walk requires no global IDs in the network, and little state at each node — at most, if no other routing mechanism is available in the network, each intermediate node may need to store the query ID and the ID of the neighbor from which it was received in order to route the query response back to the sink. If the query consists of several subqueries, then random walks can also permit the partial resolution of the query as it moves through the network, picking up pieces of the information along the way.

© 2005 by Chapman & Hall/CRC

Random walk techniques have been proposed and explored recently as a technique for load-balancing in sensor networks. Servetto and Barrenechea [3] present and analyze biased random walks to route information from a given source and destination in the network, with the goal of spreading the communication load as evenly as possible across all nodes in a grid-deployed network. The study focuses on identifying the correct probabilities for randomly forwarding the packet at each node in the network, depending on the network size, the location of that node, and the origin and destination nodes. The active querying problem, clearly, is different because there is no well-defined origin and destination, but this study illustrates one positive quality of random walks, i.e. they can provide load and energy balancing within the network.

However, the *principal* advantage of a random walk search as opposed to flooding the query is that it generally requires significantly (often orders of magnitude) fewer transmissions. It should be noted that these savings in energy come at the expense of greater response latency, which must be acceptable for random walks to make sense. This advantage has motivated the use of random-walk-based search techniques in unstructured peer-to-peer networks [4], and the rumor routing [5] and active querying in sensor networks (ACQUIRE) schemes [6] for querying sensor networks. We shall discuss each in detail, identifying some of the drawbacks of the basic random walk and how these are solved in the proposed techniques.

43.2.1 Search in Unstructured Peer-to-Peer Networks

Lv *et al.* [4] consider the problem of searching for named resources in peer-to-peer networks. In particular, they consider decentralized, *unstructured* peer-to-peer networks such as Gnutella. Such networks have much in common with sensor networks. There is no central directory where the information is stored, and there is no special topology or structured placement of data to make it easier to locate. Thus, although there have been some proposals for structured data placement in sensor networks (e.g. using geographic hash tables [7] or hierarchical multi-resolution structures [8]), most envisioned architectures for sensor networks are also unstructured. The search for a particular resource/file in a UP2P network is analogous to a one-shot query in sensor networks; therefore, the search techniques studied by Lv *et al.* [4] are pertinent to this discussion.

Lv *et al.* [4] note that random walks greatly reduce messaging cost, but at the expense of greater delay. This delay, of course, depends on the degree of replication of the data being requested. For example, if there is only one node in the network containing the resource or file being searched for, then the random walk may take unreasonably long to find the resource. Therefore, Lv *et al.* [4] also consider different replication strategies to decrease the delay (as we shall see, the rumor routing technique for sensor networks also effectively performs replication in order to increase the time to success for random walks). In addition, Lv *et al.* [4] examine the impact of increasing the number of simultaneous random walkers. As Lv *et al.* [4] point out "the expectation is that k walkers after T steps should reach roughly the same number of nodes as 1 walker in kT steps. Therefore by using k walkers we expect to cut the delay down by a factor of k." While this potentially solves the problem, it raises another: When should the multiple queries be terminated if they do not succeed? There are two solutions: one is to define a strict lifetime [time-to-live (TTL) field], after which they terminate automatically; the other is to check back periodically with the originating node to see if any of the other walks have terminated successfully. While the latter solution is found to be well suited for Internet-based UP2P networks, this kind of explicit coordination will result in additional communication overhead, which should be avoided as far as possible in sensor networks because of its energy costs.

43.2.2 Rumor Routing

Rumor routing [5] is a technique for sensor networks that is similar in spirit to searching in unstructured peer-to-peer networks. The queries are assumed to be of the type "Has event X occurred anywhere?", i.e. for a specified event in the network at an unknown location. Rumor routing includes

© 2005 by Chapman & Hall/CRC

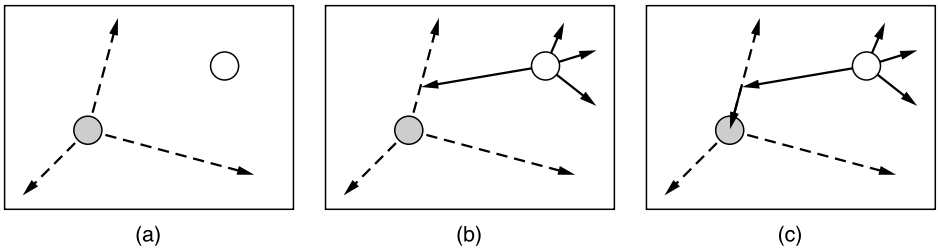


Figure 43.1. Schematic illustration of rumor routing: (a) agents creating event paths; (b) active queries from sink; (c) active query routed to pertinent node after intersection with event path.

two pieces of information: the creation of event paths (which point to the location where the event occurred) and active queries which search for event paths. Figure 43.1 shows a simple schematic illustration of this process. A node (shaded circle) which wishes to advertise an event sends out event agents; these create a routing entry in the nodes they pass through, leaving a pointer to the event. The entry may simply consist of the previous node that the agent visited on its way from the event. The event agents execute a random walk through the network. Later, when a node (unshaded circle) wishes to query for that event, it sends out its own active query, which is also as a random walk. When the active query encounters an event path, it is routed along that path to the event. At this point, the information requested about the event could be routed back to the querying node (along the reverse path, for example).

Braginsky and Estrin [5] also present several simple extensions to improve the performance efficiency of this basic scheme. For one, it permits a path agent for an event Y that encounters information about another event X along the way to pick up that information and continue on to provide information about the aggregate path to both events. This can minimize the number of event agents that are generated. Event agents may also optimize performance by taking into account shorter paths to the same event that they may encounter along the way. Because of the wireless channel, event agents can be overheard by neighboring nodes, thus leaving a thicker trail. The event agents may be generated deterministically or probabilistically, depending on whether one or more nodes detect an event. Tunable parameters for rumor routing include the average number of agents generated for a given event. To minimize looping, both the event agents and the active queries in the rumor routing scheme use TTL fields, but recently visited nodes also keep track of the agent and query IDs, so that the agents and the active queries can avoid visiting them. In simulation studies, Braginsky and Estrin [5] find that, for a range of query-to-event ratios, rumor routing significantly outperforms query flooding (which is useful when there are many events and few queries) and event flooding (which is useful when there are many queries for few events), i.e. the two basic schemes underlying the pull and push forms of directed diffusion.

For one-shot querying, the nodes along the event paths may be given summarized information about the event. Thus, one way to view rumor routing is that it improves the response time for a random walk by replicating the information sought. Braginsky and Estrin [5] mention that Monte Carlo simulations show that the probability of two randomly placed lines intersecting in an arbitrary rectangular region is as high as 69% — this translates to a greater than 99% probability that five random lines (corresponding to event paths) are intersected by a random line (corresponding to the active query). This highlights the efficiency of using event paths to improve the performance of random-walk-based active queries.

43.2.3 ACQUIRE

So we have seen that random-walk-based active queries can significantly reduce the communication overhead of query flooding. But it is certainly worth asking if this is always the case. Are there conditions where something like flooding can be beneficial? Is there some strategy between the two extremes of flooding and carrying out a random walk that makes sense under different conditions?

© 2005 by Chapman & Hall/CRC

Query flooding, as we discussed earlier, makes sense when there are few queries and if their cost can be amortized over the data that are being routed back from the source(s) to the sink, i.e. for long-standing flows. However, even if we restrict ourselves to one-shot queries, flooding can be a reasonable solution if the queries are repeated and caching can be employed successfully. In other words, the first time a query is launched it is flooded and the response for a query is cached. If there are additional queries for the same content, then it can be answered directly from the sink's cache instead of triggering a search within the network. The cost of the initial flood can then be amortized over several identical queries. This, of course, assumes that the cached data are still valid, highlighting the fact that the querying strategy might depend on the ratio of the validity time of cached data and the time between queries. These issues are explored by Sadagopan *et al.* [6], who propose a novel technique for active querying known as ACQUIRE.

The active query in ACQUIRE proceeds as follows: the querying node first checks its cache and if necessary requests an update from all nodes that are within d hops. It uses the information obtained to partially resolve the query if possible; if it still remains unsolved, it forwards it to a random node that is another d hops away. This node receiving the forwarded query (referred to as the active node) then checks its own cache, does a d-hop look ahead update, and forwards if necessary. The three-step process (examine cache, request update through local d-hop flood, forward query) is repeated until the query is fully resolved. The most important point to note about ACQUIRE is that the d-hop look-ahead allows it to be tuned flexibly. When $d = 0$, we have a pure random walk strategy; when d is comparable to the diameter of the network, it is the same as a query flood. There is a trade-off for different values of the look-ahead parameter: when the value of d is small the query needs to be forwarded more often, but there are fewer update messages at each step. When d is large, fewer forwarding steps are involved, but there are more update messages at each step. Given this trade-off, we may inquire as to what determines the optimal choice of d in terms of minimizing the total number of transmissions in the network.

Sadagopan *et al.* [6] present a mathematical model for the transmission cost of ACQUIRE as a function of d and the parameter c, which represents the expected number of updates per query. If $c = 0.01$, for example, an update is required at an active node only once in 100 queries (i.e. the information requested can be answered from the cache); if $c = 1$, caching is useless, as updates are required every query.

Let σ be the expected minimum number of random nodes that must be queried in order to resolve a query (for randomly located data, this is independent of the topology or the particular details of the search). Then it is shown [6] that E_{avg}, the expected number of transmission required to query for the information and obtain a response back to the sink using ACQUIRE with a d-hop look-ahead is given as

$$E_{\text{avg}} = \begin{cases} \sigma[f(d)]^{-1}[c(f(d) + g(d) + 2d] & d \geq 1 \\ 2\sigma & d = 1 \end{cases} \tag{43.1}$$

where $f(d)$ is the expected number of nodes within d hops of an active node and $g(d)$ is the expected number of messages required for all nodes within d hops to respond to a query from an active node.

It turns out that, depending on the value of c, E_{avg} will always increase with respect to d (in which case $d = 0$, and a random walk is the best strategy), will decrease first and then increase with respect to d (in which case some intermediate value of d represents the best strategy), or will decrease with respect to d (in which case a large d corresponding to flooding is the best strategy). As expected, when c is high and close to unity, i.e. when data dynamics are high and caching is of no benefit, a random walk is the best choice. When c is very low, close to zero, cached data remain valid for a longer time and flooding can actually be beneficial. Figures 43.2 and 43.3 show numerical plots of the performance of ACQUIRE for a square-grid topology (each node neighbors four nodes), assuming $\sigma = 100$.

43.3 Active Queries with Direction

While random walks are a simple choice for active querying, they are essentially a form of blind search and can be improved upon by incorporating additional knowledge. In some cases it may be desirable to

© 2005 by Chapman & Hall/CRC

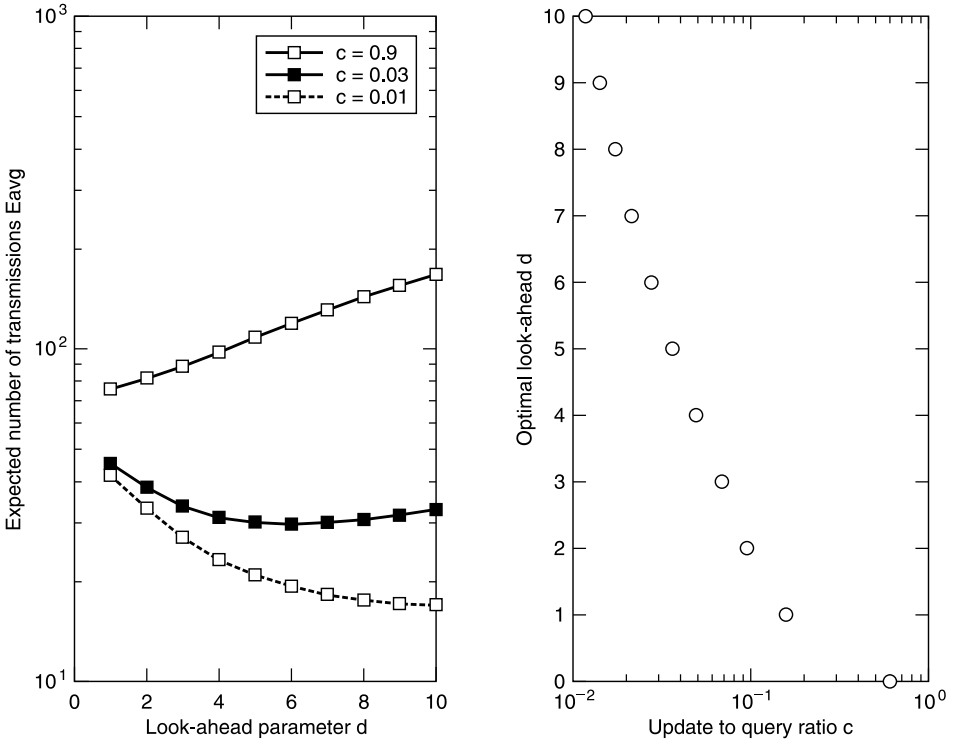


Figure 43.2. Performance curves for ACQUIRE. The plot on the left shows the transmission cost as a function of the look-ahead parameter for different update-to-query ratios, and the plot on the right shows how the optimal look-ahead varies with this parameter.

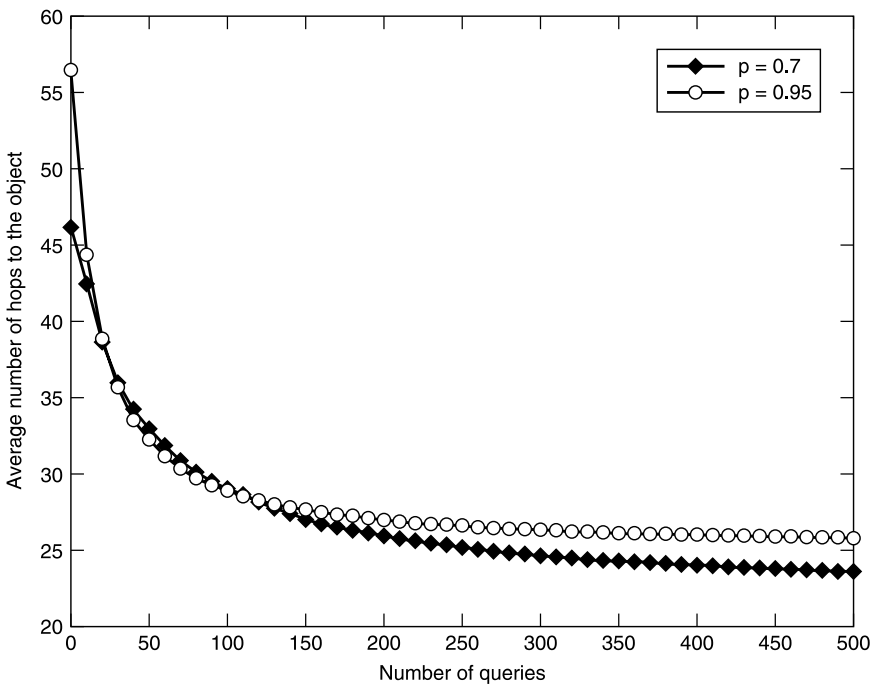


Figure 43.3. Performance of LEQS showing decrease in query cost over time, depending on the learning rate.

© 2005 by Chapman & Hall/CRC

send the queries on a predetermined trajectory, which can be done through source routing or using the routing on curves technique [9]. In some cases it may be possible to route the queries to the locations where they are most likely to be answered, based on the past history of similar queries, as exemplified by the learning-based efficient querying of sensor networks (LEQS) technique [10]. In yet other scenarios, it would be helpful to exploit the query semantics and the spatial correlation of information available in network nodes to direct the query to its intended location, as exemplified by geographic forwarding techniques [11] and information-driven sensor querying IDSQ/constrained anisotropic diffusion routing (CADR) [12]. We shall now describe these techniques in turn.

43.3.1 Source Routing and Routing on Curves

In some situations it will be desirable to send active queries through the network to sweep through a particular set of nodes, to undertake a prespecified trajectory. In traditional networks, one technique for doing this is source routing [13], in which the sequence of nodes to be visited is included in the packet header. In sensor networks, particularly those where fine-grained localization is available, a new technique has been proposed [9] to send queries on geographically specified trajectories. With this technique, called routing on curves, an arbitrary spatial trajectory is described in the active query packet and, at each step, the node that receives this packet forwards it to the neighbor whose location most closely matches the described trajectory. For example, this trajectory could be a straight line (useful, for example, in rumor routing), and each node that receives the packet would forward it in the desired direction to the neighbor which is closest to the described line. Assuming fine-grained geolocation information is available for all nodes, and assuming that nodes are located with sufficient density, this technique can approximate arbitrary trajectories very closely.

43.3.2 LEQS

Consider one-shot queries for the location of an identifiable object in the sensor field, i.e. an object which has a regular pattern of location that can be described with a probability distribution. For example, this could be a target that is known to be always near one of three sensor locations with corresponding probabilities p_1, p_2, or p_3. If this distribution is stationary, then there is the possibility of improving the efficiency of repeated queries for this object through learning. One mechanism that has been proposed to incorporating learning to improve the efficiency of active queries is LEQS [10].

LEQS is a simple localized and distributed learning algorithm for active querying. In this algorithm, sensor nodes maintain weights indicating the probability with which a given active query is forwarded to each neighbor. The query response is used to update these weights on the reverse path, effectively *training* the network to locate the object more and more efficiently over time.

Upon node deployment and setup, each sensor node identifies its immediate neighbors and sets up a vector of weights (one for each identifiable queried object A) in a querying table. Weight $W_{i,j}$ represents the probability that a query for object A that arrives at node i will be forwarded to node j. Initially, if a given node has k neighbors, each neighbor of the node is assigned an equal weight of $1/k$. Each active query starts from the sink and, with the probabilities denoted by the weights at each node, will be forwarded randomly from node to node until the object is located. Thus, initially, when the node weights are all equal, this is equivalent to the unbiased random walk. A backtracking technique is incorporated to prevent looping. Once the object is located, the response of the query is then sent back directly to the sink on the reverse path of this active query (using the local state maintained at each node about where it received the query from), and this is when the weights are updated. Each node i on the reverse path increases the weight of its next hop h_i (i.e. the node to which it had forwarded the message). The query response on the return path contains a counter that is incremented hop by hop, so that all nodes on the reverse path get an indication of the number of hops d_i that they were from where the query terminated successfully. This information is used in the weight update rule, which is described below.

© 2005 by Chapman & Hall/CRC

Each node on the reverse path first calculates a learning factor

$$L_i = p d_i^{-\alpha} \tag{43.2}$$

In this expression, p (in $[0, 1]$) and α (greater than zero) are both learning parameters that determine the rate of learning, as well as the dependence on the distance to query termination. If h_i is the node from which node i received the query response (i.e. its next hop neighbor) and $N(i)$ is the set of all neighbors of that node, then the weights at node i are updated as follows:

$$\begin{cases} W_{i,j}(t+1) = W_{i,j}(t)(1-L) & \forall j \in N(i) \backslash h_i \\ W_{i,h1}(t+1) = W_{i,h1}(t) + L \sum W_{i,j}(t) & \forall j \in N(i) \backslash h_i \end{cases} \tag{43.3}$$

Over time (repeated queries) this learning policy trains the network to forward active queries efficiently towards locations where they are most likely to be resolved. Figure 43.3 shows how the query cost (measured in number of hops to locate the object, i.e. till query response) decreases over time with LEQS, depending on the learning rate parameters. While a low learning rate results in slow convergence, a learning rate that is too high may result in faster convergence but to a higher query cost solution.

43.3.3 Geographic Forwarding

The performance of active queries can also be improved by incorporating query semantics and exploiting spatial structure. A simple example of this is the use of geographic forwarding techniques, an excellent survey of which is provided by Mauve *et al.* [11]. Such techniques are very useful for one-shot queries with geographic scoping (e.g. "what is the temperature at location (x, y)?"). The basic idea is to forward such a query at each step to a neighbor in the general direction of the destination. As pointed out by Mauve *et al.* [11], there are several variations of greedy forwarding techniques. These include the MFR technique [13], in which the packet is forwarded to the neighbor that is closest to the destination, and the NFP technique [14], in which the packet is forwarded to the nearest neighbor with forward progress (in order to minimize contention). Since greedy forwarding techniques are susceptible to local minima or voids where no forward progress can be made to the destination, it is possible to use the perimeter routing mode incorporated into the greedy parameter stateless routing (GPSR) scheme [15].

43.3.4 Sensing Driven Querying

Active queries in geographic forwarding can be conceived as a form of greedy search that is trying to minimize the objective function of distance to the destination. A generalization of this is to consider arbitrary objective functions that depend on sensor readings. For example, queries such as "where is the target located?" may search to locate the node which maximizes the corresponding sensor reading (this could be a signal strength measurement). Because sensor readings are spatially correlated due to the underlying physical phenomena, a greedy search that follows gradients in the network can often provide efficient active query performance. A scheme that utilizes such information is the IDSQ/CADR mechanism detailed in [12] (also described earlier in this book).

IDSQ provides a mechanism for sensor selection to maximize the information gain (balanced with its communication costs), while CADR is essentially an active querying mechanism to determine how to route the queries dynamically through the network. It uses a composite objective function that incorporates both information gain and communication costs and routes the query at each step by choosing the neighbor that lies in the direction of the gradient of this objective function. Routing the query in this way is shown to maximize the information obtained while minimizing the communication costs. Techniques such as IDSQ/CADR demonstrate the ability to incorporate query semantics to decide the trajectory of active queries on-the-fly, for even greater efficiency.

© 2005 by Chapman & Hall/CRC

43.4 Conclusions

We have described a class of active querying mechanisms that are suited for one-shot/instantaneous queries. Active querying mechanisms, which are essentially mobile-agent-based search techniques, are particularly useful because they require orders of magnitude less communication than query flooding. As we saw, the simplest active query mechanisms involve the use of random walks. The trade-off is that random walks result in high latency for response, particularly when the information queried is sparsely located in the network. Solutions for this include the use of multiple simultaneous walks and replication of the data, e.g. through the setting up of event paths in rumor routing. We also found that, depending on the ratio of updates to queries and the availability of caching, hybrid schemes such as ACQUIRE, which provide for random walks with limited flooding for local updates, can be more efficient.

We also examined several mechanisms to improve over random walks by introducing direction to the active queries. This can be done explicitly through source routing or by prespecifying the trajectory using routing on curves. LEQS shows how the notion of query direction can be learned over time by exploiting history, if the queried object has an unchanging (or slowly changing) location pattern. Finally, queries can be directed to terminate quickly based on the query semantics by exploiting geographic and sensor information, such as in GPSR and IDSQ/CADR.

While the existing literature has presented preliminary work and suggested possible design principles, there is still considerable room for future work on developing intelligent active queries for different scenarios. Active query mechanisms will clearly form a useful part of the portfolio of querying techniques that will be deployed in practical sensor networks.

References

[1] Govidan, R. *et al.*, The sensor network as a database, USC CS Technical Report 02-771, 2002.

[2] Bonnet, P. *et al.*, Querying the physical world, *IEEE Personal Communications*, 7, 10, 2000.

[3] Servetto, S.D. and Barrenechea, G., Constrained random walks on random graphs: routing algorithms for large scale wireless sensor networks, in *Proceedings of 1st ACM International Workshop on Wireless Sensor Networks and Applications*, Atlanta, GA, September 2002.

[4] Lv, Q. *et al.*, Search and replication in unstructured peer-to-peer networks, in *Proceedings of ACM International Conference on Supercomputing*, 2002.

[5] Braginsky, D. and Estrin, D., Rumor routing algorithm for sensor networks, in *Proceedings of 1st ACM International Workshop on Wireless Sensor Networks and Applications*, Atlanta, GA, September 2002.

[6] Sadagopan, N. *et al.*, The ACQUIRE mechanism for efficient querying in sensor networks, in *Proceedings of IEEE International Workshop on Sensor Network Protocols and Applications (SNPA'03)*, Anchorage, AK, May 2003.

[7] Ratnasamy, S. *et al.*, GHT — A geographic hash-table for data-centric storage, in *Proceedings of 1st ACM International Workshop on Wireless Sensor Networks and Applications*, Atlanta, GA, September 2002.

[8] Ganesan, D. *et al.*, DIMENSIONS: why do we need a new data handling architecture for sensor networks? in *Proceedings of First Workshop on Hot Topics in Networks (HotNets-I)*, Princeton, NJ, October 2002.

[9] Nath, B. and Niculescu, D., Routing on a curve, in *Proceedings of First Workshop on Hot Topics in Networks (HotNets-I)*, Princeton, NJ, October 2002.

[10] Krishnamachari, B. *et al.*, LEQS: learning-based efficient querying for sensor networks, USC Computer Science Technical Report CS 03-795, 2003.

[11] Mauve, M. *et al.*, A survey on position-based routing in mobile *ad hoc* networks, IEEE Network Magazine, 15(6), 30, 2001.

[12] Chu, M. *et al.*, Scalable information-driven sensor querying and routing for *ad hoc* heterogeneous sensor networks, *International Journal of High Performance Computing Applications*, 16(3), 293, 2002.

© 2005 by Chapman & Hall/CRC

[13] Takagi, H. and Kleinrock, L., Optimal transmission ranges for randomly distributed packet radio terminals, *IEEE Transactions on Communications*, 32(3), 246, 1984.
[14] Hou, T.C. and Li, V.O.K., Transmission range control in multihop packet radio networks, *IEEE Transactions on Communications*, 34(1), 38, 1986.
[15] Karp, B. and Kung, H.T., GPSR: greedy parameter stateless routing for wireless networks, in *Proceedings of ACM MOBICOM 2000*, Boston, MA, August 2000.

© 2005 by Chapman & Hall/CRC

VII

Self-Configuration

44. Need for Self-Configuration *R.R. Brooks* *847*
Problem Statement • Top-Down Control • Bottom-Up Reconfiguration • Self-Organization Models • Summary • Acknowledgments

45. Emergence *R.R. Brooks* *855*
Problem Statement • Continuous Models • Discrete Models • Characterization of Pathological Behavior • Summary • Acknowledgments and Disclaimer

46. Biological Primitives *M. Pirretti, R.R. Brooks, J. Lamb, and M. Zhu* *863*
Background • Tools Used • Ant Pheromone Model • Summary • Acknowledgment and Disclaimer

47. Physics and Chemistry *Mengxia Zhu, R.R. Brooks, Matthew Pirretti, and S.S. Iyengar* *879*
Introduction • Discussion of Two Examples from Physics and Chemistry • Modeling Tool • Idealized Simulation Scenario • Applying Physics and Chemistry to WSN Routing • Protocol Comparison and Discussion • Acknowledgment and Disclaimer

48. Collective Intelligence for Power-Aware Routing in Mobile *Ad Hoc* Sensor Networks *Vijay S. Iyer, S.S. Iyengar, and N. Balakrishnan* *895*
Introduction • Artificial Ants for Routing • Results • Conclusion

49. Random Networks and Percolation Theory *R.R. Brooks* *907*
Notation • Background • Graph Theory • Erdös–Rényi Graphs • Small-World Graphic • Scale-Free Graphs • Percolation Theory • *Ad Hoc* Wireless • Cluster Coefficient • Mutuality • Index Structure • Graph Partitioning • Expected Number of Hops • Probabilistic Matrix Characteristics • Network Redundancy

© 2005 by Chapman & Hall/CRC

and Dependability • Vulnerability to Attack • Critical Values • Summary • Acknowledgments and Disclaimer

50. **On the Behavior of Communication Links in a Multi-Hop Mobile Environment** *Prince Samar and Stephen B. Wicker*..... 947
Introduction • Related Work • Link Properties • Simulations • Applications of Link Properties • Conclusions • Appendix 50A

Autonomous sensor networks are distributed amorphous computing environments, consisting of a large number of unreliable nodes and communications links, subject to intermittent failure, likely destruction, and limited power resources. Adaptation to chaotic conditions, such as those faced by autonomous sensor networks, is best performed by self-organizing systems. This chapter concentrates on the organization of self-configuring distributed sensor networks.

Brooks proposes the top-down control and bottom-up reconfiguration and self-organizing models to produce deliberative adaptation of the network to overcome the complexity inherent in real time environments. Surveillance networks are subject to intermittent failure and require bottom-up emergent control push interacting with the user's pull and top-down control.

Brooks focuses on *emergence*, the major concept in self-organization. Emergent phenomena are behavior observed in macroscopic systems that are an indirect consequence of the microscopic behavior of individual participants. The author emphasizes on continuous models, discrete models, finding the pathologies inherent in a given system, finding concrete examples of pathologies in existing data, and recognizing emerging pathologies online to correct problems as they arise.

Pirretti focuses on biological primitives for studying and designing applications for wireless sensor networks (WSNs). He discusses how to utilize these concepts from biology to design distributed routing algorithms for WSNs that have the following characteristics: (i) low resource consumption, (ii) highly adaptive and near-optimal routing, and (iii) high resilience to various error conditions. He provides an in-depth case study of one particular biological primitive, ant pheromone, which was used to develop a routing algorithm in a WSN for a Military Operations in Urban Terrain (MOUT) application. He also provides explanation of the tools used, parameter settings, optimizations, performance, and power consumption.

Zhu *et al.* focuses on Ising model of statisitical physics, diffusion limited aggregation, spin glass model, spin glass simulation results, and multi-fractal for solving routing problems in WSNs. WSNs operating in an urban terrain are subject to unpredictable topological disturbances. They analyze the approaches for: (*i*) ability to adapt, (*ii*) robustness to internal errors, and (*iii*) power consumption, and focus on comparisons to currently prevailing *ad hoc* WSN routing techniques.

Iyer *et al.* focus on collective intelligence for power aware routing in mobile *ad hoc* sensor networks. Emergent behavior gives rise to collective intelligence or the swarm intelligence, which has been used here to identify routes in mobile *ad hoc* sensor networks. In mobile *ad hoc* sensor networks the problem manifolds in complexity due to mobility of the nodes. The authors have used specialized packets known as ants to establish routes. They also address the issue of scarcity of energy in mobile devices by making the algorithm power-aware.

Brooks focuses on random network formalisms for designing, modeling, and analyzing survivable sensor networks. He emphasized on how to use random graph formalisms for wired and wireless P2P systems. He also explains how to estimate network performance, network dependability, network redundancy, system dependability, system phase changes, vulnerability to intentional attack, whether or not the network is connected, and the expected number of hops between nodes. He uses concepts from graph theory, linear algebra, and probability.

Samar and Wicker describe the behavior of individual links in multi-hop systems. These individual behaviors combine to create the statistical landscape, which the self-organization behaviors attempt to contain.

In summary, this section focuses on designing, modeling, and analyzing of self-organization in DSNs.

© 2005 by Chapman & Hall/CRC

44

Need for Self-Configuration

R.R. Brooks

44.1 Problem Statement

Surveillance networks can be viewed at many levels of abstraction. Effective systems must adapt at each level. These networks will be composed of a large number of autonomous, possibly heterogeneous, devices. At the highest level of abstraction, all devices must work in a coordinated manner as if they were a single entity. User interests vary dynamically. Top-down control is needed to insure that tasks performed by the network fulfill operational requirements. Specific regions and/or target types will be of interest sometimes, but not at others. Connectivity maintenance requires high-level coordination.

Bottom-up self-organization is necessary to guarantee the system's ability to adapt to unforeseen events. The environment varies dynamically. Weather, wind direction, and ambient noise change the effective sensing ranges [1]. All nodes have finite energy resources and eventually cease to function. Radio communications can be jammed. These factors require bottom-up reaction and reconfiguration in a fully autonomous mode.

Surveillance networks are distributed amorphous computing environments, consisting of a large number of nodes and communications links, subject to intermittent failure, likely destruction, and limited power resources [2]. Human configuration and control of the entire system is futile. Humans steer the network by declaring general interests. Translating higher level tactical objectives into low-level activities is an ill-posed problem. Efficient resolution of this problem in a chaotic network of failure-prone nodes will require bottom-up emergent-control *push* interacting with the user's *pull*.

It is important to decentralize the system without sacrificing robustness and accuracy. Top-down control is need driven and deliberate. Bottom-up control is reactive and geared towards maintaining an orderly substrate on top of an underlying chaotic environment. The goal is to produce deliberative adaptation of the network to overcome the complexity inherent in real environments.

44.2 Top-Down Control

Scalable top-down control requires a hierarchical topology. Flat organizations need direct communication with all participating nodes. Once the number of components surpasses a critical value, the communications overhead becomes untenable. Similarly, the latency imposed by the volume of

© 2005 by Chapman & Hall/CRC

information that users must process in a large, flat hierarchy makes decisions from higher levels obsolete by the time they can be formulated.

Within a hierarchy, each layer has its own responsibilities and information requirements. The top level provides general guidance. Each level in the hierarchy accepts commands from the higher level, refines them to fit local constraints, creates more detailed instructions, and distributes them to lower levels. Events occurring at lower levels are relayed to higher levels, which extract relevant information for their superiors, thereby reducing the volume of data. Higher level instructions should not be overly restrictive to allow freedom for quick reaction as needed. This is an ill-posed problem [3], since conflicts and contradictions may arise as instructions percolate through the system. It can be expressed as a hierarchy of control languages, or interacting automata, and implemented as an adaptive hierarchy of discrete-event dynamic-system (DEDS) controllers [3,4].

Robust control methods extend continuous control, enabling construction of plant controllers able to perform correctly and predictably within known noise bounds [5]. Given these known bounds, hybrid control methods can switch from robust controller C_1 to C_2 when the underlying dynamics of the system move into a mode of operation that is unstable for C_1 but not C_2 [6]. DEDS controllers exist either as the higher level switching logic used by hybrid controllers, or as methods for controlling plants whose underlying dynamics are unknown or unknowable [3,4].

DEDS control is amenable to the construction of hierarchical control structures, since usually only the dynamics of the lowest level in the control hierarchy could possibly be modeled using continuous variables and differential equations. Higher levels function by manipulating discrete variables indicating events occurring and behaviors active in the system. Two tools dominate for expressing and designing DEDS controllers: finite state automata (FSA) [4] and Petri nets [7].

FSA have the advantage of translating control strategies into formal languages. The language defines an abstract grammar for constructing strings using an alphabet defined by individual events. We have developed tools for analyzing system controllability and robustness by analyzing the formal language generated by the FSA. When this is extended to hierarchical control, each layer in the hierarchy works as a Mealy machine that does three things: (i) absorbs events generated by immediate superiors and subordinates; (ii) modifies its internal state; (iii) generates new events to be absorbed by its immediate superiors and subordinates. We have used this approach in DARPA-sponsored research exploring air-campaign planning and execution. Figure 44.1 shows the suppression of an enemy air defenses air-operations mission simulation using our hierarchy of discrete event controllers.

The approach shown in Figure 44.1 uses FSA controllers for each entity in the hierarchy. Adding a stack to the local controllers would allow context-free grammars, like modern programming languages, to be recognized. Adding a tape would allow each participant to execute any computable function [8]. On the other hand, the FSA-based control hierarchy effectively defines a type of cellular automaton (CA) variant. Since CAs of a given size are capable of performing arbitrary computations [9], the use of FSA is not particularly constraining.

Petri nets are advantageous for modeling resource contention and synchronization [10]. We have used extensions of Petri nets to model complex military operations [11]. A Petri net is constructed using modular components to design an operation. In order to verify that the system is free of deadlock and livelock conditions, a Karp–Miller tree is constructed. The Karp–Miller tree is an FSA, where each state is defined by the number of tokens in each place of the Petri net [5]. This state definition format is consistent with the construction of vector controllers [12,13]. Figure 44.2 shows a simple Petri net and its associated Karp–Miller tree.

In this manner, DEDS FSA controllers can be automatically derived from Petri nets. The Petri net construct can guarantee the lack of deadlock and livelock. This design defines behaviors triggered in response to discrete events inferred from locally available information (including transmissions from other participating sensor nodes). The number of events considered is typically small (less than 100) and, therefore, the bandwidth consumed by control messages is small. However, behaviors triggered by the DEDS control hierarchy can consist of sophisticated continuous control regimes or optimization methods [14]. Making decisions with local information, when possible, saves power and bandwidth.

© 2005 by Chapman & Hall/CRC

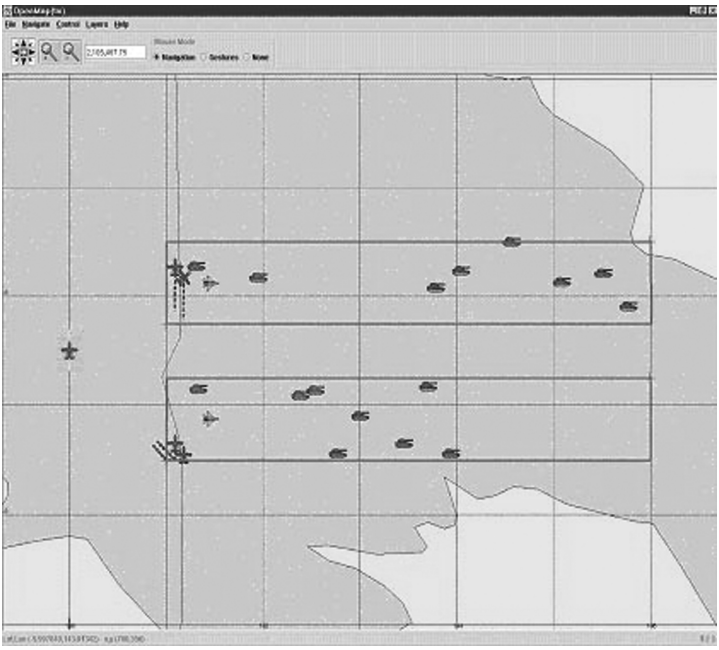


Figure 44.1. Air campaign simulation using C4iSim. Forces are controlled by a DEDS hierarchy.

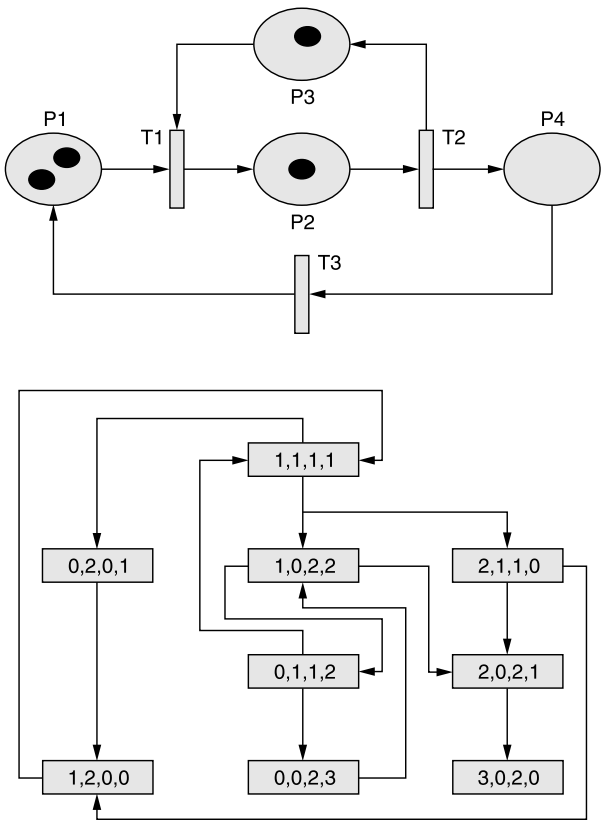


Figure 44.2. Top: Petri net for a limited resource problem. Bottom: its associated Karp–Miller tree.

© 2005 by Chapman & Hall/CRC

The role of an individual node may vary over time. For example, if a supervisory node fails then it should be possible to elect a replacement. We implemented this approach for networks of underwater oceanographic sampling robots [15] by requiring that all nodes have identical software. We have implemented a more sophisticated approach for sensor network tasking in the DARPA SenseIT program [16,17]. Mobile code is used to overcome the storage constraints that are ubiquitous in embedded systems. Nodes are initially configured with behaviors they are most likely to need. As needs change, programs are downloaded and storage reclaimed. In many ways, the local software configuration is managed like a local cache. We have implemented polymorphism and distributed dynamic linking to support this in networks of heterogeneous nodes. Control roles in the DEDS hierarchy may change to suit environmental conditions better.

The system must be robust. Its continued survival depends greatly on power conservation. Node longevity requires:

- Classification and detection decisions made as close to the sensor as possible.
- Network traffic routed in a manner that reduces redundancy.
- During periods when little of interest occurs, nodes enter a state of reduced power consumption.
- Node movement and beamforming occur rarely.

On the other hand, system robustness and operational considerations sometimes require:

- Critical information transmitted redundantly, taking independent paths through the network. This reduces single points of failure.
- Critical information made available to operational personnel with minimal latency.
- Information veracity verified through comparison with readings of multiple sensing modalities.
- Information sharing for beamforming and other expensive analysis techniques.
- Power expenditure for node movements to change fields of regard or communications paths.

44.3 Bottom-Up Reconfiguration

Equally important is the system's ability to respond autonomously to perceived changes in its operational environment. Self-configuration and adaptability are essential, since the nodes must adapt to maintain three separate hierarchies:

- *Control hierarchy*. The control hierarchy must be able to morph and re-emerge, since connectivity may be broken at any point in the hierarchy.
- *Sensor coverage*. Changes in the environment (such as fog) or failures in the sensor network can result in the occlusion, or loss of coverage, of regions of interest. It may be necessary to reassign sensors or move nodes to re-establish coverage.
- *Network connectivity*. In addition to observing regions of interest, the network must report events to data consumers. To respond to jamming, or low battery power, the network configuration must be flexible. Multiple routes within the network should be maintained. It may be necessary for nodes to reposition themselves autonomously.

Since connectivity can be broken at any point in the hierarchy, self-organization must be supported to allow the hierarchy to re-emerge. Candidate approaches to self-organization have been derived by researchers in artificial life studying insect colony coordination [15], in physics studying quantum optics [19], and chemistry studying nonequilibrium thermodynamics [20].

Each DEDS controller defines its local behaviors in response to an uncontrollable and frequently hostile environment. Tasks are performed in a distributed environment requiring coordination among multiple nodes. Communication may be corrupted with noise. Components are prone to failure. As network size increases, the probability of all components functioning at a given time decreases exponentially [21]. Nonlinear interactions exist between failure modes. Under these conditions, efficient

© 2005 by Chapman & Hall/CRC

operation cannot rely on static plans. Van Creveld [22] has defined five characteristics that hierarchical systems need to adapt to this type of chaotic environment [23]:

- Decision thresholds far down in the hierarchy.
- Self-contained units at a low level.
- Bottom-up and top-down information circulation.
- Informal communications.
- Commanders seek to supplement routine reports.

Organizations using this approach have been successful in market economies, war, and law enforcement [24]. Our approach is influenced by, and is consistent with, these points. Each entity is self-contained and makes as many decisions as possible locally. Operational information and data travel in both directions in the hierarchy. The final two points imply informal activities that are difficult to implement in automated systems. In this section, we discuss local decisions made far down in the hierarchy to counteract the chaotic environment. Self-organization needs to occur to allow the system to adapt quickly to arbitrary unforeseen events.

The distributed control architecture can be modeled as a network of interacting automata. This is an abstraction of distributed systems that is a modification of CAs [25]. CAs are synchronously interacting sets of elements (network nodes) defined as abstract machines. A CA is defined by:

- d the dimension of the automaton
- r the radius of an element of the automaton
- δ the transition rule of the automaton
- s the set of states of an element of the automaton

An element's (node's) behavior is a function of its internal state and those of neighboring nodes as defined by δ. Wolfram's study of CAs [9] has found four qualitative classes of CA:

- Stable — where all elements eventually evolve to the same state.
- Periodic — where the global system evolves into regular structures.
- Chaotic — where no discernible structure emerges and the system remains in a constant state of flux.
- Interesting — where no global stable regime occurs but areas of local stability emerge spontaneously.

Chaotic regimes are clearly undesirable. Stable and periodic regimes are desirable, but in some conditions they may be inflexible and incapable of adapting to a changing environment. The interesting regime allows for spontaneous adaptation. It has been hypothesized that the ideal state for adaptation is stability on the edge of chaos [26], where chaos supports a quick search through new regimes when stability is no longer possible.

A DEDS model can produce a hierarchy of interacting automata that coordinate the overall system. This can be studied as a hierarchy of CAs of increasingly fine (coarse) resolution as traversed from the root (leaves) to the leaves (root). Inputs to each automaton at each time step are its own state and the perceived states of its neighbors. For low levels of the hierarchy, the automata state includes the state of the environment in the region covered. Given the chaotic and uncontrollable nature of the underlying dynamics, it is likely that the regime of the lowest level is, and will remain, chaotic. Higher levels in the hierarchy should, however, retain a stable regime no matter what happens at the lowest levels of the control hierarchy. It is hypothesized that middle levels of the hierarchy should adapt, forming a bridge between the top-level stability and lower level chaos.

The communications and sensing coverage constructs are just as important as the control hierarchy. Since they depend both on system state and node position, they cannot be adequately modeled using a pure CA approach. Recent work [27] has adapted concepts developed as part of theoretical explanations of self-organization of laser beams [19] to explain similar problems in urban planning. The model of free agents in a cellular space [27] allows agents to migrate between elements in a CA environment.

© 2005 by Chapman & Hall/CRC

The behaviors of agents and CA elements influence each other. In our approach, agents are sensing platforms that may or may not be mobile. CA elements are grid points defining surveillance regions. Surveillance targets can also be modeled as agents. In this way, we can study self-organization regimes for sensor coverage and maintenance of communication connectivity. We currently plan on only studying first-order effects. It may in fact be possible to study interactions between control hierarchy, sensor coverage, and communications connectivity maintenance, but this depends on the results of the modeling thrust.

44.4 Self-Organization Models

Many theories exist for self-organization and spontaneous emergence of order [18,19,25,26,28]. Those based on thermodynamic models [19,20] are likely to be untenable, since they are based on models defined by sets of differential equations. It is unlikely that we will be able to derive suitable sets of equations to adequately represent general circumstances. Some artificial life constructs [18,25] are based on a few basic interactions among multiple participants:

- Positive feedback — includes recruitment and reinforcement of behaviors.
- Negative feedback — counterbalances positive feedback to stabilize the system.
- Amplification of fluctuations — randomness and fluctuations are crucial to system adaptation.
- Multiple interactions — simple behaviors at the microscopic level can provide intelligent adaptations at the macroscopic level.

These interactions form the basis of pheromone interactions used by insects for foraging and nest building, as shown in Figure 44.3 [18,25]. This approach can be seen as a way of constructing distributed feedback loops. It has been used in developing a number of transport and communications applications [29]. Adaptation of these concepts to adaptation in sensor networks will be described in detail in Chapter 46 of this book.

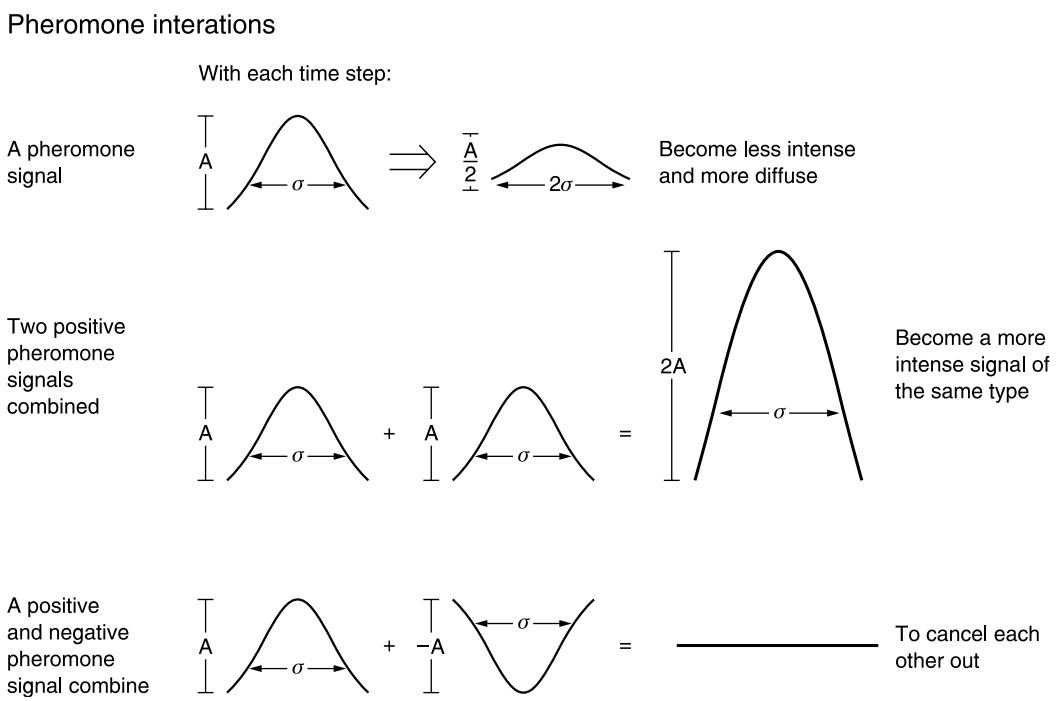


Figure 44.3. The pheromone abstraction provides a convenient way of expressing time-dependent positive and negative distributed feedback systems.

© 2005 by Chapman & Hall/CRC

Insect colonies use pheromones to provide positive and negative feedback signals [18]. This approach has been applied to distributed route planning [18] and to military command and control [30]. We use pheromone mechanisms to encode information needed for coordination by the components in this mechanism. Other explanations of self-organization look at genetic constructs [26]. This concept is appealing, since DNA organization can be seen as a probabilistic grammar of an alphabet defined by a small number of amino acids. The use of abstract languages, to define order, parallels the FSA models of discrete event control. CAs model the distributed and synchronous nature of the problem. Synthetic pheromones provide primitives for communication and coordination. The network then formulates its own solution to the problem via local interactions.

44.5 Summary

This chapter discussed the organization and control problems inherent in sensor networks. Each network is composed of multiple semi-autonomous nodes embedded in a chaotic environment. The system as a whole has to cooperate and provide information about the environment. Each individual node needs to react immediately to inputs from the environment. There is a natural tension between top-down control, as will be discussed in Chapter 51, and bottom-up adaptation, as discussed in this section. Similarities exist between this conflict and the nature of military command and control. Technical details about implementing self-organizing systems and distributed control systems are provided in the other chapters.

Acknowledgments

This effort is sponsored by the Defense Advance Research Projects Agency (DARPA) and the Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-99-2-0520 (Reactive Sensor Network). The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the author's and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), the Air Force Research Laboratory, or the U.S. Government.

References

[1] Swanson, D.C., *Signal Processing for Intelligent Sensor Systems*, Dekker, San Francisco, 2000.
[2] Brooks, R.R. and Iyengar, S.S., *Multi-Sensor Fusion: Fundamentals and Applications with Software*, Prentice Hall, 1998.
[3] Phoha, S. *et al.*, A constructivist theory for distributed intelligent control of complex dynamic systems, in *Symposium on Advances in Enterprise Control*, San Diego, November 1999.
[4] Rudie, K. and Wonham, W.M., Think globally, act locally: decentralized supervisory control, *IEEE Transactions on Automatic Control*, 37(11), 1692, 1992.
[5] Zhou, K. *et al.*, *Robust and Optimal Control*, Prentice Hall PTR, Upper Saddle River, NJ, 1996.
[6] DeCarlo, R.A. *et al.*, Perspectives and results on the stability and stabilizability of hybrid systems, *Proceedings of the IEEE*, 88(7), 1069, 2000.
[7] Brooks, R.R. *et al.*, Stability and controllability analysis of fuzzy Petri net JFACC models, in *DARPA–JFACC Symposium on Advances in Enterprise Control*, November 15–16, 1999.
[8] Hopcroft, J.E. and Ullman, J.D., *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, MA, 1979.
[9] Wolfram, S., *Cellular Automata and Complexity*, Addison-Wesley, Reading, MA, 1994.
[10] David, R. and Alla, H., *Du Grafcet aux Reseaux du Petri*, Hermes, Paris, 1989.
[11] Adaptive C2 coalitions air campaign model, Penn State ARL Technical Report delivered to DARPA JFACC Program office and AFRL, December 1999.

© 2005 by Chapman & Hall/CRC

[12] Li, Y. and Wonham, W.M., Control of vector discrete event systems I — the base model, *IEEE Transactions on Automatic Control*, 38(8), 1214, 1993.
[13] Li, Y. and Wonham, W.M., Control of vector discrete event systems II — controller synthesis, *IEEE Transactions on Automatic Control*, 39(3), 512, 1994.
[14] Phoha, S. *et al.*, Tactical intelligence tools for distributed agile control of air operations, in *2nd Symposium on Advances in Enterprise Control*, Minneapolis, July 2000.
[15] Phoha, S. *et al.*, A high fidelity AOSN simulator for intelligent control of networked ocean sampling assets, *IEEE Journal of Oceanic Engineering*, 26(4), 646, 2001.
[16] Brooks, R.R. *et al.*, Reactive sensor networks: mobile code support for autonomous sensor networks, in *Distributed Autonomous Robotic Systems DARS 2000*, Springer Verlag, 2000.
[17] Brooks, R.R., Distributed dynamic linking, Penn State Invention Declaration, May 2000.
[18] Bonabeau, E. *et al.*, Self-organization in social insects, in *Working Papers of the Satna Fe Institute 1997*, http://www.santafe.edu/sfi/publications/Working-Papers/97-04-032.txt, 1997 (last accessed on 7/24/2004).
[19] Haken, H., *Synergetics*, Springer Verlag, Berlin, 1983.
[20] Nicolis, G. and Prigogine, I., *Self-Organization in Non-Equilibrium Systems*, Wiley, New York, 1977.
[21] Siewiorek, D.P. and Swarz, R.S., *The Theory and Practice of Reliable System Design*, Digital Press, Bedford, MA, 1982.
[22] Van Crefeld, M.L., *Command in War*, Harvard University Press, Cambridge, MA, 1986.
[23] Czerwinski, T., *Coping with the Bounds: Speculations on Nonlinearity in Military Affairs*, National Defense University, Washington, DC, 1998.
[24] Cebrowski, A.K. and Garsta, J.J., Network-centric warfare: its origin and future, in *Proceedings of the Naval Institute*, January 1998, http://www.usni.org/Proceedings/Articles98/PROcebrowski.htm (last accessed on 7/24/2004).
[25] Adami, C., *Introduction to Artificial Life*, Springer Verlag, New York, 1998.
[26] Kaufmann, S., *The Origins of Order*, Oxford University Press, New York, 1993.
[27] Portugali, J., *Self-Organization and the City*, Springer Verlag, Berlin, 2000.
[28] Brooks, R.R., Stigmergy — An intelligence metric for emergent distributed behaviors, in *NIST Workshop on Performance Metrics for Intelligent Systems*, Gaithersburg, MD, August 2000.
[29] Dorigo, M. *et al.*, The ant system: optimization by a colony of cooperating agents, *IEEE Transactions on Systems, Man, and Cybernetics — Part B*, 26(1), 29, 1996.
[30] Parunak, H. and Brueckner, S., Synthetic pheromones for distributed motion control, in *JFACC Symposium in Advances Enterprise Control*, November 1999.

© 2005 by Chapman & Hall/CRC

45
Emergence

R.R. Brooks

45.1 Problem Statement

Surveillance networks should be dependable entities composed of multiple, unreliable sensor nodes. This is a distributed amorphous computing environment consisting of a large number of nodes and communications links subject to intermittent failure, likely destruction, and limited power resources. Human configuration and control of this environment is futile. Humans steer the network by declaring general interests. Translating higher level tactical interest into low-level activities is an ill-posed problem requiring a bottom-up emergent-control *push* to interact with the user's *pull.* This system is a large complex entity whose behavior is defined by interactions among multiple participants.

The national infrastructure now consists of dynamic systems relying on interactions among multiple semi-autonomous entities. The physics of these types of system has been studied and understood in continuous [1–3] and discrete [4–7] domains. Hybrid systems that integrate both are needed and have been neglected. Traditional dependability models rely on queuing theory and statistical methods [8–10]. Since many system pathologies are caused by dynamic nonlinear interactions between components [11], it is essential to create these hybrid mathematical models and update control theory to provide a new generation of systems that adapt to chaotic environments.

Among the common characteristics of engineering designs are failure modes and pathologies. Kott and Krogh [11] provide an initial taxonomy of pathologies for complex systems, which lists pathological behaviors and mechanisms that are often responsible. Pathologies are general descriptions of ways the system deviates from its goal states. The taxonomy includes ubiquitous problems, such as race conditions, deadlock, livelock, oscillation, and thrashing. Reasons for pathology occurrence include lack of information, inappropriate thresholds, inconsistency, and excessive constraints [11]. In this section we describe the relevant mathematical models of complex systems that express their underlying structure and pathologies.

Pathological behavior is endemic to complex systems, which are composed of multiple nontrivial components with nonlinear dynamic interactions. Components may not be homogeneous. Their internal composition may vary over time. Studies of interacting components indicate that extremely complex chaotic behavior occurs even in systems consisting of homogeneous, simple automata with static configurations [4]. System behavior has both local and global aspects. Correct system behavior relies on synergetic effects emerging between local and global regimes.

© 2005 by Chapman & Hall/CRC

Military logistics is an excellent example of a complex system. System inputs and outputs are material supply and demand, both of which are uncertain. This uncertainty cannot adequately be modeled as a stochastic variable, since many aspects are subject to influence by a hidden, pernicious, and intelligent adversary. Transport and storage is subject to failure and confusion. In addition, the overall system has uncertainties in sensing and actuation, commonly referred to as the *fog of war* and *friction* [12].

Advances in computing and mathematics now make it possible to study and model the interactions and dynamics of large complex systems that were undecipherable until recently. Haken [2] and Nicolis [3], in physics and chemistry, have determined principles of self-organization in matter that hold for complex systems in biology [13] and sociology [14]. Applications of these principles have been found in diverse domains ranging from lasers [2] to explaining behavior of insect colonies [15].

Systems are modeled as macroscopic entities made up of a large number of smaller elements [2,14]. Self-organization is found when complex macroscopic behaviors occur as a nontrivial consequence of interactions between the individual elements. Self-organization occurs only in systems far from equilibrium [3]. System behavior ranges from strict order to chaos.

We use the term emergent to characterize system behaviors with the following characteristics:

- They arise from interactions between multiple entities.
- All behaviors are based on local information.
- They contain a significant stochastic component.
- Positive feedback is used to encourage desirable behaviors.
- Negative feedback stabilizes the system.
- Global system behaviors contain spontaneous phase changes.
- The global behaviors are not obvious consequences of the local behaviors.

45.2 Continuous Models

The concept of emergence can be applied to many existing engineering systems, and others currently being designed. The fact that recent studies of Internet traffic detect self-similarity in traffic flow [16,17] supports our supposition that the basic tenets of synergetics [2] will hold for critical infrastructures like sensor networks. In these engineered systems, a large number of complex individual entities merge to form a larger system. In many ways, the macroscopic behavior of the aggregate system emerges from interactions between individuals. These behaviors limit the ability to control the system from the top down in ways that are not immediately evident. An example of this is fault propagation through complex coupled systems producing unforeseen macroscopic errors. These errors are difficult to foresee and/or correct, like the failure of the electric infrastructure in the western United States in the summer of 1996 [18,19].

For emergent behavior to arise, the systems must be complex and have a large number of degrees of freedom. System behavior depends on a large number of system variables and has many modes of behavior. The effects of any given variable may be controllable or unstable, depending on the mode of operation [2]. The change from one mode of system behavior to another occurs abruptly at a critical value of a single system variable [20].

Variables may be continuous or discrete. For continuous variables, a singular value decomposition (SVD) can be performed to decouple interactions and find those variables with the strongest influence on system behavior. The effects of some variables are controllable, whereas nonlinear interactions make the effects of variations of other variables inherently unstable and unpredictable. The set of controllable variables derived through SVD thus provides a system with greatly reduced dimensionality.

At this point, the *slaving principle* can be used to provide structure in an otherwise chaotic system [2]. This states that, although the system as a whole is not entirely controllable, the controllable variables can be used to steer the system's evolution. These controllable variables provide bounds for the chaotic actions forced by the unstable variables. The unstable variables are considered slaves of the controllable ones. This control system can be used until another critical value is reached and the dominant mode of

© 2005 by Chapman & Hall/CRC

operation changes abruptly. It can even be used to force the system past another critical value. Methods for deriving and finding critical values, where the behavior of physical systems changes abruptly, are given by Jensen [20]. This is an extension of the basic concepts of robust control, where control strategies are shaped around instabilities and sources of error [21].

A matrix system representation can capture many essential system characteristics. If internal component structure is ignored, transportation models can be used. Material suppliers are sources y and end users are sinks u. Transportation paths and storage depots are represented as limited-capacity edges that form a network connecting sources to sinks. This problem can be formulated in terms of graph theory, linear programming, nonlinear differential equations [22,23], or control theory [21].

In terms of graph theory, there are many methods for designing systems that maximize flow through a network. In addition to algorithms specifically designed for flow maximization, dynamic programming is a common approach. In the case of stochastic systems, methods generally attempt to optimize the expected flow through the system. These models require knowledge of statistical distributions of the stochastic components, information consistency, and known functions that express supply and demand. These models are too static and deterministic. Pathologies will arise, but the model cannot express how or why.

One way of designing the system would be to try to have the supply match the demand, which could be phrased as a control system:

$$x' = Ax + Bu \quad y = Cx + Du \quad \text{or} \quad \begin{bmatrix} x' \\ y \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} = G \begin{bmatrix} x \\ u \end{bmatrix}$$

where x is the set of internal state variables, y is the supply, and u is the demand. One then attempts to derive matrices A, B, C, and D to make y follow u. In the linear case, it is possible to solve the equations to optimize a set of known criteria. It is even possible to use stochastic noise variables and define a structured singular value, where the system remains stable and controllable as long as noise remains within limits [21]. Well-defined measures of stability, controllability, observability, and robustness exist in control theory [21].

The matrix formalism is still applicable when the system is nonlinear, even though its application becomes significantly more complex [24]. Note that in analyzing these control systems the SVD is used to extract the principle components of the system. Singular values (eigenvalues) λ_i express the modes inherent in the system. Sorting the values in order of magnitude allows a threshold T to be set. Singular values larger than the threshold ($\lambda_i > T$) are considered significant and retained. Those smaller than the threshold ($\lambda_i < T$) are not significant and are discarded. This often allows the dimensionality of the problem to be reduced significantly.

Military and social systems are notoriously nonlinear in nature [25], with an unknown number of higher order interactions. Hagen and Wunderlin [26] express dynamic nonlinear systems with multiple operational modalities. These modalities are expressed by system singular values λ_i. Any operation mode is a function of interactions between these modes. The interactions can lead to instability, oscillations, chaotic dynamics, and most other pathologies. Any modality can be approximated by a linearization of the system using a matrix of the λ_i. Different control parameters are relevant for different system modalities. In particular, many system parameters become irrelevant, or subservient to another parameter, in specific behavior modes [26]. This can drastically reduce the number of degrees of freedom of a system, as long as the underlying dynamics are known.

Although control theory is useful for expressing and discovering many pathologies, it still has significant limitations. Unfortunately, in military logistics, noise inputs are not adequately modeled as stochastic variables. They are at least partially controlled by the plans of an intelligent adversary. Inconsistencies in the system may be partially due to pilferage and/or sabotage. The confusion caused by chaotic conditions also leads to difficulties in collecting information and executing plans [12]. Internal system variables are subject to inconsistency and corruption. Many significant aspects of the plant are unknown and possibly even unknowable [12].

© 2005 by Chapman & Hall/CRC

Alternatively, the network can be approached as a nonequilibrium system [3]. Many biological and chemical systems exist where microscopic flux and chaos are offset by macroscopic order. In this model, stable macroscopic regimes emerge and provide a predictable system as long as system variables remain within certain limits. When critical values of system parameters are reached, the internal structure of the system changes radically. In many cases these critical values are due to internal thresholds and effects of positive feedback. This is the basis of many fractal and chaotic interactions seen in nature [20]. Internal thresholds and positive feedback are also present in logistics systems. In the example of medical supplies, when more than a certain percentage of people contract an illness the illness is more likely to spread. This type of positive feedback relation is hidden from a straightforward control model. Results from nonequilibrium systems are consistent with the pathology taxonomy of Kott and Krogh [11]. Many pathologies are due to incorrect threshold values in a system. These threshold values are critical values where the system modifies its control regime. Jensen [20] discusses methods for deriving and verifying critical values where system behavior changes radically.

45.3 Discrete Models

Kauffman [27] explores the relation between higher order interactions and system adaptation. He finds that, in complex systems, ordered regimes are frequently separated by chaotic modes that adapt quickly to find new ordered regimes. His work applies differential equations, discrete alphabets, and cellular automata (CAs) to system analysis. His approach is consistent with the economics study of Chang and Harrington [28], the urban planning study of Portugali [14], and the air operations study of Brooks [7]. These studies all explicitly consider differences in information availability to decision makers in distributed systems. Independent entities interact within a distributed space. It appears that discrete systems and CAs are a reasonable approximation of the underlying continuous dynamics [27]. It is also not necessary for all the underlying dynamics to be understood or modeled [14,27]. For this reason, it appears that a model of agents in a cellular space is suited for expressing pathologies for systems with unknown underlying dynamics. It is also suited for expressing the interaction between global and local factors.

Some variables may be innately discrete, especially in engineered systems. In other cases, the dynamics may not be sufficiently known (or measurable) for models using differential equations to be practical. It is advisable to model these systems using CAs abstraction [4,5]. CAs of sufficient size have been shown capable of performing general computation [5]. A CA is a synchronously interacting set of elements (network nodes) defined as a synchronous network of abstract machines [6]. A CA is defined by:

- d the dimension of the automaton
- r the radius of an element of the automaton
- δ the transition rule of the automaton
- s the set of states of an element of the automaton

An element's (node's) behavior is a function of its internal state and those of neighboring nodes as defined by δ. The simplest instance of a CA is uniform, has a dimension $d = 1$, a radius $r = 1$, and a binary set of states. In this simplest case, for each individual cell there are a total of 2^3 possible configurations of a node's neighborhood at any time step. Each configuration can be expressed as an integer v:

$$v = \sum j_i \times 2^{i+1}$$

where i is the relative position of the cell in the neighborhood (left$=-1$, current position $= 0$, right $= 1$), and j_i is the binary value of the state of cell i. Each transition rule can therefore be expressed as a single integer r:

$$r = \sum_{v=1}^{8} j_v \times 2^v$$

© 2005 by Chapman & Hall/CRC

where j_v is the binary state value for the cell at the next time step if the current configuration is v. This is the most widely studied type of CA. It is a very simple many-to-one mapping for each individual cell. The aggregated behaviors can be quite complex [5]. Wolfram's work has found four qualitative equivalence classes for CAs [4]:

- *Stable* — evolving into a homogeneous state.
- *Repetitive* — evolving into a set of stable or periodic structures.
- *Chaotic* — evolving into a chaotic pattern.
- *Interesting* — evolving into complex localized structures.

These states are useful for modeling complex systems. Under normal operating conditions, stable or repetitive behavior is desirable. When the system has been pushed out of equilibrium, chaotic and interesting interactions are desirable. They enable the system to explore the set of possible adaptation, quickly, eventually finding a new configuration adapted to the new environment [27].

The agents in a cellular space model [14] augments CAs by adding agents that are defined as abstract automata. The agents migrate within the cellular space. Their behavior depends on their own state, the state of cells in their neighborhood, and possibly on a small number of global variables.

45.4 Characterization of Pathological Behavior

The question remains as to how to find the pathologies inherent in a given system, find concrete examples of pathologies in existing data, and recognize emerging pathologies on-line to correct problems as they arise. Starting from the pathology taxonomy of Kott and Krogh [11], it is possible to analyze systems to find likely examples as long as internal dynamics, control signals, and stochastic distributions are known [21]. This is not the case. The problem is a difficult pattern recognition application of looking for amorphous patterns in a poorly defined space.

Where dynamics are known, continuous models of the known subproblems will be constructed. As with control and nonequilibrium systems, singular values will be computed for the continuous models. This extracts the most influential modes of operation and discards those with little impact [24,26]. Further, constructing hierarchies within the remaining control parameters can find those that are slaves to other parameters in given operational regimes [26]. This reduction of the number of degrees of freedom greatly simplifies pattern recognition and decreases sensitivity to noise [29].

Where dynamics are unknown, a model of agents in a cellular space [14] can be established. The model expresses the decision makers in the system, their interactions, and their environment. Continuous problems can be subsumed into the cellular space or agent as appropriate. Comparing the dynamics of this model with the dynamics recorded in databases of logistics scenarios allows for correction of modeling areas and derivation of statistical distributions for stochastic aspects of the problem. It may or may not be possible to model antagonistic aspects of the system satisfactorily.

Given historical data, an initial starting point is to create datasets that express scenarios in which a particular pathology occurs. Ideally, the singular values derived for the well-known continuous modeled subproblems would adequately describe the system as a whole. Using the singular values would then reduce the number of degrees of freedom in the pattern recognition problem and allow self-organizing memory to partition the system [14,26]. Adaptive resonance theory [29] and Kohonen maps [26] would be suited to creating systems that recognize emerging pathologies if this is the case.

If classification using the singular values of subsystems whose dynamics are well known is insufficient, then data mining approaches may be needed to infer additional diagnostic information [30–32]. This augments the original information with a number of rough set and fuzzy association rules. Interestingly, work with latent semantic analysis has shown that the SVD can be used to infer concepts and groupings from text data. Therefore, it should be possible to unify the semantic information in linguistic variables with the singular values from systems with known dynamics in a common vector space and use principal components analysis to derive a set of control variables of minimal size. Mined rules can influence the system globally, local system cells, or the agents in the cellular space.

© 2005 by Chapman & Hall/CRC

45.5 Summary

This chapter provided general background on the concept of emergence. This background should be useful in understanding some of the other chapters in this section of the book. Emergence is a powerful concept that helps explain many natural phenomena. Natural systems adapt around disturbances and quickly find a new equilibrium. Biological systems, particularly insect colonies, are also capable of finding near-optimal solutions to complex problems with no centralized control. The groundbreaking work introduced here has helped explain how self-organization works in the natural world. The chapters on biological, chemical, and physics metaphors to self-organization in this section provide technical details on how we are adapting these concepts to engineering systems.

Acknowledgments and Disclaimer

This work was supported by the Defense Advanced Research Projects Agency (DARPA), and administered by the Army Research Office under ESP MURI Award No. DAAD19-01-1-0504. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the Defense Advanced Research Projects Agency (DARPA) and Army Research Office.

References

[1] Alligood, K.T. *et al.*, *Chaos: An Introduction to Dynamical Systems*, Springer Verlag, New York, 1996.
[2] Haken, H., *Synergetics: An Introduction*, Springer-Verlag, Berlin.
[3] Nicolis, G. and Prigogine, I., *Self-Organization in Non-Equilibrium Systems*, Wiley & Sons, New York, 1977.
[4] Wolfram, S., *Cellular Automata and Complexity*, Addison-Wesley, Reading, MA, 1994.
[5] Delorme, M., An introduction to cellular automata, in *Cellular Automata: A Parallel Model*, Delorme, M. and Mazoyer, J. (eds), Kluwer Academic Publishers, Dordrecht, 1999, 5.
[6] Sarkar, P., A Brief History of Cellular Automata, *ACM Computing Surveys*, 32(1), 80, 2000.
[7] Brooks, R., Stigmergy—an intelligence metric for emergent distributed behaviors, in *NIST Workshop on Performance Metrics for Intelligent Systems*, Gaithersburg, MD, August 2000, 1.
[8] Siewiorek, D.P. and Swarz, R.S., *The Theory and Practice of Reliable System Design*, Digital Press, Maynard, MA, 1982.
[9] Rai, S. and Agrawal, D.P. (eds), *Distributed Computing Network Reliability*, IEEE Computer Society Press, Los Alamitos, CA, 1990.
[10] Rai, S. and Agrawal, D.P. (eds), *Advances in Distributed System Reliability*, IEEE Computer Society Press, Los Alamitos, CA, 1990.
[11] Kott, A. and Krogh, B., Toward a catalog of pathological behaviors in complex enterprize control systems, in *Proceedings from November 1999 DARPA-JFACC Symposium on Advances in Enterprize Control*, San Diego, CA, November 15–16, 1999, 1, http://www.darpa.mil/iso/jfacc/symposium/sess2-1.doc (last accessed on 8/16/2004).
[12] Van Creveld, M.L., *Supplying War: Logistics from Wallenstein to Patton*, Cambridge University Press, Cambridge, UK, 1980.
[13] Eigen, M., Ursprung und Evolution des Lebens auf molekularer Ebene, in *Evolution of Order and Chaos*, Haken, H. (ed.), Springer-Verlag, Berlin, 1982, 6.
[14] Portugali, J., *Self-Organization and the City*, Springer Verlag, Berlin, 2000.
[15] Bonabeau, E. *et al.*, Self-organization in social insects, *Working Papers of the Santa Fe Institute 1997*, http://www.santafe.edu/sfi/publications/Working-Papers/97-04-032.txt, 1997 (last accessed on 8/16/2004).

© 2005 by Chapman & Hall/CRC

[16] Leland, W.E. *et al.*, On the self-similar nature of ethernet traffic (extended version), *IEEE/ACM Transactions on Networking*, 2(1), 1, 1994.
[17] Grossglauser, M. and Bolot, J.-C., On the relevance of long-range dependence in network traffic, *IEEE/ACM Transactions on Networking*, 7(5), 629, 1999.
[18] CNN Online, "Domino Effect" zapped power in west, August 11, 1996, http://www.cnn.com/TECH/9608/11/power.outage/index.html (last accessed on 8/16/2004).
[19] Online Newshour, Blackout, July 3, 1996, http://www.pbs.org/newshour/bb/science/blackout_7-3.html (last accessed on 8/16/2004).
[20] Jensen, H.J., *Self-Organized Criticality*, Cambridge University Press, Cambridge, U.K., 1998.
[21] Zhou, K. *et al.*, *Robust and Optimal Control*, Prentice Hall, Upper Saddle River, NJ, 1996.
[22] Haberman, R., *Mathematical Models: Population Dynamics, and Traffic Flow*, SIAM, Philadelphia, 1998.
[23] Smulders, S.A., *Control of Freeway Traffic Flow,* CWI Tract, Amsterdam, 1991.
[24] Brogan, W.L., *Modern Control Theory*, Prentice Hall, Upper Saddle River, NJ, 1991.
[25] Czerwinski, T., *Coping with the Bounds: Speculations on Nonlinearity in Military Affairs*, National Defense University, Washington, DC, 1998.
[26] Hagen, H. and Wunderlin, A., Application of synergetics to pattern formation and pattern recognition, *Self-Organization*, *Emerging Properties and Learning*, NATO ASI Series B: vol. 260, Plenum Press, NY, 1991, 21.
[27] Kauffman, S.A., *The Origins of Order: Self-Organization and Selection in Evolution*, Oxford University Press, Oxford, U.K., 1993.
[28] Chang, M.-H. and Harrington, J.E., Centralization vs. decentralization in a multi-unit organization: a computational model of a retail chain as a multi-agent adaptive system, in *Working Papers of the Santa Fe Institute, 1999.*
[29] Pandya, A.S. and Macy, R.B., *Pattern Recognition with Neural Networks in C++*, CRC Press, Boca Raton, FL, 1996.
[30] Tsaptsinos, D., Rough sets and ID3 rule learning: tutorial and application to hepatitis data, *Journal of Intelligent Systems*, 8(1–2), 203, 1997.
[31] Tsumoto, S., Knowledge discovery in medical databases based on rough sets and attribute-oriented generalization, in *The 1998 IEEE International Conference on Computational Intelligence*, 1998, 1296.
[32] Tsymbal, A. *et al.*, Advanced dynamic selection of diagnostic methods, in *11th IEEE Symposium on Computer-Based Medical Systems*, 1998, 50.

© 2005 by Chapman & Hall/CRC

46
Biological Primitives

M. Pirretti, R.R. Brooks, J. Lamb, and M. Zhu

46.1 Background

46.1.1 Characteristics of Biological Primitives

Biological primitives are models of biological systems that exhibit the property of self-organization. These systems consist of numerous, usually homogeneous, biological entities interacting with each other with relatively simple behaviors in such a way as to exhibit global behaviors that are too complex to be exhibited by the individual entities. The individual entities interact utilizing only local information, and they lack any sort of ''master plan'' or centralized leadership. The literature from biology refers to this appearance of global behavior from the interaction of numerous entities exhibiting simple behaviors as emergence. For further information regarding self-organization, see Chapters 44 and 49; for more information on emergence see Chapter 45.

Naturally occurring self-organizing systems are not unique to the realm of biology; there are also physical and chemical self-organizing systems where the individual entities are inanimate. An example of such a system would be the emergence of patterns in a sand dune due to the interactions amongst individual grains of sand [1]. There are two important distinctions that should be made in regards to physical systems and biological systems with self-organizing properties. First, individual units in biological systems tend to be much more complicated than those from physical systems, as biological organisms represent more complexity than do inanimate objects. The second difference is that in a chemical or physical self-organizing system the individual units only follow the laws of physics, whereas biological systems additionally exhibit behaviors that are the result of an ever-changing genetic makeup. It is this second difference that makes biological primitives of particular interest to researchers; with time the system will fine-tune its behavior through the process of natural selection, whereas with chemical and physical systems there is no evolution, and the system will follow the same rules for all time [1]. A further in-depth discussion of physical and chemical self-organizing systems is provided in Chapter 47.

There are two different mechanisms by which self-organizing systems control themselves: positive and negative feedback. Positive feedback reinforces desirable behaviors by making it increasingly attractive to exhibit certain behaviors; negative feedback limits positive feedback so that the positive

© 2005 by Chapman & Hall/CRC

feedback will not get out of control. This is done by making it increasingly unattractive to keep performing the desired behavior. The net result is that a global behavior emerges from the interplay of positive feedback, negative feedback, and the local behaviors of the entities in the system.

46.1.2 Why Apply it to Wireless Sensor Networks

Perhaps the best way to describe why biological primitives are applicable in the realm of wireless sensor networks (WSNs) is to first explain some of the difficulties that arise in WSNs. Our WSN application is for use in military operations in urban terrain (MOUT) scenarios. In these urban scenarios, radio communications are impeded by various obstructions which may cause the shortest path between two points to not be a straight line. The chaotic nature of MOUT missions causes unreliable and highly variable paths, causing only transient paths to exist. In spite of these difficulties, timely communications are required for the application to be useful. This is compounded by the fact that the global behavior of having an effective WSN routing methodology is extremely complicated. Further, traditional packet-routing techniques have extremely poor performance, due to the assumptions that error rates at the link level are infrequent and that duplicate or missing packets are caused by congestion. Additionally, centralized control of this type of network is undesirable and unrealistic due to reliability, survivability, and bandwidth considerations, whereas distributed control does not have these issues [2]. Distributed control also has other advantages: (i) increased stability by avoiding single points of failure; (ii) simple node behaviors replace complicated global behaviors; (iii) enhanced responsiveness to changing conditions, since nodes react immediately to topological changes instead of waiting on central command.

The nature of these challenges makes it attractive to use a model based upon self-organizing biological systems to solve the inherent difficulties associated with WSNs; a biological-based model would be intrinsically distributed, would use positive and negative feedback based upon local information to attain the desired global behavior of routing, and could react very rapidly to transient activities that are common in a MOUT scenario by reacting immediately at a local level in lieu of waiting upon central command to determine the proper course of action to be carried out. The behaviors of the nodes in a biological system tend to be relatively simple, making design and verification straightforward once implemented in a WSN application. Further, with biological self-organizing systems there is the ability of the system to evolve with time, and this could be potentially exploited to have a system which can tune itself to enhance performance over time.

46.1.3 Exemplar Models from Biology

For didactic reasons, several examples of self-organizing biological systems are provided. The first example, *Dictyostelium discoideum*, a type of amoeba, explains the complex spiral pattern that these creatures form under specific circumstances. The second example, ant pheromone, explains how certain species of ant use pheromones to maintain short paths between their nest and food sites.

46.1.3.1 *Dictyostelium discoideum*

One interesting example of self-organizing systems is that of a certain species of cellular slime mold, amoebas, called *D. discoideum*. When there is a steady supply of food each *D. discoideum* acts independently of its neighbors, eating and then splitting after it has eaten an ample amount of food. This process continues until the food supply is diminished and the individual *D. discoideum* amoebas begin to aggregate together, forming complex spiral-like patterns.

The mechanism by which the amoebas form these patterns is based upon the secretion of an attractant known as cAMP. The amoebas secrete this substance during the starvation period every 5 to 10 min, while they move towards higher concentrations of cAMP. If the amoebas sense a cAMP pulse that is of a large enough magnitude then they will secrete an even larger quantity of cAMP, causing local hot spots where numerous amoebas have conglomerated. Eventually, as the concentration of cAMP

© 2005 by Chapman & Hall/CRC

increases, the amoebas start to become desensitized to this attractant, and consequently they will curtail their cAMP production.

In this example, the global behavior of having the amoebas form complicated spiral patterns emerges from the individual behaviors of the amoebas, which are utilizing local information (the density of cAMP) as the only means to shape their behaviors. The cAMP attractant is acting as positive feedback for forming the global pattern, and the amoebas' eventual desensitization to cAMP acts as negative feedback by limiting the production of cAMP. For further information on the mechanisms of pattern formation in *D. discoideum*, see Camazine *et al.* [1].

46.1.3.2 Ant Pheromone

A second example from biology that exhibits self-organization is the method by which certain species of ant, for instance *Lasius niger*, the black garden ant, forage for food. Ants attempt to find paths between their nest and potential food sources. They release two different pheromones: (i) search pheromone when they look for food and (ii) return pheromone when they have food and are returning to the nest. It is these pheromones that help the ants navigate to where the food sources and the nest are located. Ants that are looking for food will typically follow the highest concentration of return pheromone; ants returning to the nest tend to follow the highest concentration of search pheromone. There are also scouting ants that move about randomly looking for new food sources.

The pheromones vary with time due to diffusion and evaporation. The initial pheromone laid down by the ants will begin to diffuse outwards as time progresses. Additionally, pheromone will begin to evaporate, and eventually the pheromone will have evaporated to the point where it is undetectable.

In this example, the global behavior of having the ants form short paths between their nest and various food sources emerges from the behavior of the ants and their pheromones. The positive feedback in this system is the ants laying down pheromones. Good paths, i.e. those that go directly from the food to the nest, will be traveled by more ants and will consequently receive more pheromone. Negative feedback is exhibited from how the pheromone evaporates and diffuses, preventing the pheromone from attaining unbounded concentrations.

46.2 Tools Used

Most important to selecting a particular biological model to utilize in a WSN is to ensure that the global behavior of the biological model is consistent with the desired global behavior of the WSN. If the application and the base model are drastically different it then would be quite difficult get the proper global behavior to emerge. Once a proper biological primitive has been selected it can be tailored to model the desired application better; and with this done, the choice of tools to implement a solution will dictate the rest of the solution development.

46.2.1 Cellular Automaton

A cellular automaton (CA) is an excellent manner to model a complex self-organized system. The entire model space is placed upon a grid (typically two-dimensional) and is driven in discrete time slots known as generations. Each cell in the grid is characterized as having a location defined by its coordinates in the grid and a state, which is a model-specific representation of the interaction between a cell's parameters and the environment. Interactions amongst the cells can only take place as the model iterates through each generation. The cells interact with other cells in their neighborhood, and can only change their own state based upon the state of their neighbors, their state, and a set of rules assigned to that cell.

In agent-based CAs there are also special cells called agents that can be used in the model. Agents are essentially cells that have the ability to move from cell to cell, interacting with other cells and other agents.

The power of this modeling technique is that one only has to define relatively simple rules for the cells and agents to follow and what information should be stored in the state of the cells and agents.

© 2005 by Chapman & Hall/CRC

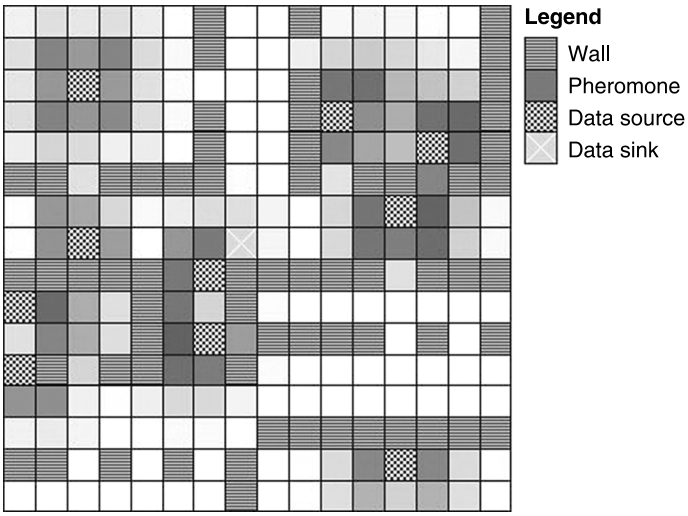


Figure 46.1. Sample trace from a model designed with Cantor.

46.2.2 Cantor

To develop our model, we have developed Cantor, which is a tool that aids in the creation of CAs.

Cantor provides a generic infrastructure that is useful for designing CAs, specifically it only requires users to provide the rules that the cells will follow and what information should be included in a cell's state. Some useful features of Cantor are that it supports any number of dimensions in the grid, it automatically creates data output files denoting the state of all of the cells for each generation, it allows the user to specify the creation of neighborhoods, and the user can specify a floorplan consistent with a MOUT application. A sample trace from a model written using Cantor is provided in Figure 46.1. In this example, the cells are clearly defined by the regularly spaced horizontal and vertical lines. Further, there are elements of a MOUT application in this example, specifically the striped gray regions. These areas are walls, where no communication is possible; for instance, wireless communication is not possible through certain materials.

46.3 Ant Pheromone Model

The in-depth ant pheromone model discussion that follows is a case study of how it can be possible to use biological primitives to solve one particular problem in WSNs. The problem that was solved was that of information routing in a MOUT application. The model is a biological primitive based on ant foraging that forms short paths to and from a particular source and destination. The model is capable of adapting rapidly to changing system conditions; hence, the manner in which ants forage for food is a logical choice for use in a WSN application.

46.3.1 How the Biological Model Was Modified

There are a few notable differences between the model that was created and the way in which ants forage for food. Most notably, we have included certain aspects to our model to study a MOUT scenario effectively. Figure 46.2 shows an idealized MOUT terrain. Walls signify any obstruction that can block radio signals. Open cells are open regions allowing signal transmission. Open doors (closed doors) are choke points for signals that periodically allow (disallow) transmission. Finally, obstructions are intermittent disturbances that occur at random throughout the sensor field. Random factors are inserted to emulate common disruptions for this genre of network. Each square capable of transmission

© 2005 by Chapman & Hall/CRC

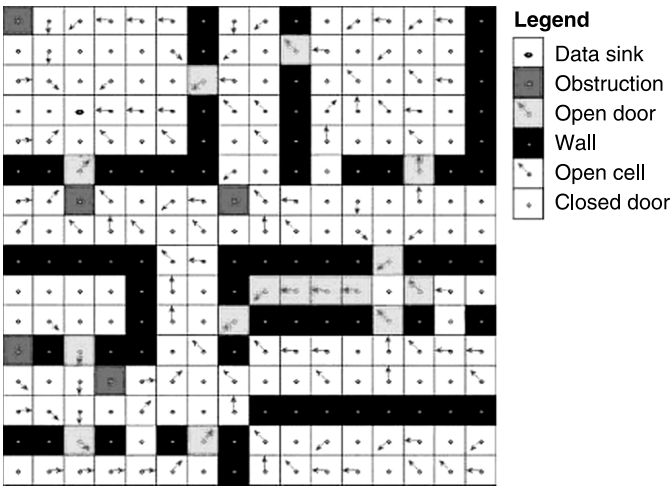


Figure 46.2. Example of a MOUT application.

contains a sensor node. This amounts to having a sensor field with a uniform density. This provides an abstract example scenario approximating situations likely to exist in a real MOUT situation.

Another difference between our application and that of the biological system is the inclusion of a communication-reduction technique known as gossip, which will be discussed later. Additionally, several principals reflect the fact that this model was specifically designed to implement a routing algorithm, which would not be particularly meaningful in the pure biological model (e.g. keeping track of path length).

46.3.2 Dorigo's Work

The work of Dorigo consisted of utilizing the foraging behavior of ants to solve various discrete optimization problems, such as the traveling salesman problem and the quadratic assignment problem [3]. Di Caro and Dorigo [4] went on to develop AntNet, an algorithm that solves the routing problem for packet switched telecommunications networks.

46.3.3 Differences from Dorigo's Work

We have utilized a system similar to AntNet, except that our system has been designed specifically for use in a WSN. This application requires the system to adapt to local transient behaviors much more rapidly than what was required from Dorigo's environment. Further, this application is based on wireless communications, which are by nature much less reliable than those found in wireline networks due to the much higher incidence of errors on the wireless link. To overcome these potential issues, the model had to be designed to be very adaptive to changing environmental factors, and be very resilient to error conditions.

46.3.4 Application to the Routing Problem

The manner in which the system was designed was by initially assuming that there are wireless sensor nodes distributed uniformly throughout the network of interest, specifically the network is initially connected. This does not mean to say that with time some of the nodes will not fail and result in disconnectivity. We have categorized the network into the following regions: (i) a region which is being covered by a sensor node that does not need to transmit or receive information; (ii) a region being covered by a sensor node (a data source) that has information to send to another node in the network; (iii) a region that is covered by a node that is the recipient of information from one or more nodes

© 2005 by Chapman & Hall/CRC

(a data sink); (iv) a region that is not being covered by a sensor node (e.g. the region may be unsecured or it may have properties that make radio transmission impossible).

Using this partitioning of the area under observation by the WSN allows the ant pheromone model to be used to set up routes running between the data sources and the data sinks, utilizing intermediate sensor nodes to pass along the information, all the while avoiding regions that for various reasons do not permit communication.

Employing the ant pheromone model will allow the network to adapt rapidly to an ever-changing substrate, while maintaining good routes between the data sources and their respective data sinks.

46.3.5 Tools Used

The Cantor tool has been utilized to design the routing algorithm. Using this tool requires the creation of rules that the cells and ants follow, and what information is contained in the state of an ant and the state of a cell. What follows is a list of each of these rules and states for the cells and ants.

The following rules control the behavior of cells:

- *Diffusion rule*. Implements the diffusion behavior of the ant pheromone, enabling pheromone from one cell to diffuse into its neighboring cells taking into account neighboring cells that may obstruct communication (e.g. walls).
- *Disturbance rule*. Allows for a cell to change its state type at any given point in the execution of the algorithm. This is primarily used to model the transient nature of the application. For instance, one could convert a cell that initially allowed communication into one that does not allow communication.
- *Evaporate rule*. Every time step a portion of the ant pheromone gets dissipated due to the process of evaporation utilizing the following equation:

$$p(t) = p(t-1)e^{-r} \tag{46.1}$$

  where $p(t)$ is the pheromone level at time t, e is Euler's constant, and r is the rate of evaporation.

- *Gossip rule*. Implements the gossip principle on the passing of pheromone information. Each generation, a cell determines whether it should take part in communicating its pheromone information. Gossip and how it relates to ant pheromone will be discussed later.
- *Haywire rule*. Allows cell malfunctions to be incorporated into the model. Cells that follow this rule select a random pheromone level instead of calculating it based upon their neighbors' state and their own state.
- *Preserve state rule*. The CA keeps track of two states for each cell and each ant, called the previous state and the current state. The previous state, which is the current state from the prior generation, is used by the ant algorithm to formulate the current state. Using the previous state is done so that each cell in the grid will be utilizing the same information when it is updating its current state, thus avoiding race condition amongst the states. The function of this rule is to copy the current state from the last generation into the previous state of the current generation.
- *Spawn rule*. This causes each ant nest to periodically spawn an ant.

The following rules were developed to control the behavior of ants:

- *Lifetime rule*. Each ant has a lifetime associated with it, which is the number of generations a particular ant has been alive. This rule increments the lifetime of each living ant once per generation.
- *Movement rule*. This rule causes the ants to follow the movement behavior that has been described above. Additionally, this rule implements the two types of error condition that the ants can exhibit, called haywire random and haywire weighted. Both error conditions cause the ants to follow some other behavior than the desired movement behavior. Haywire random is

© 2005 by Chapman & Hall/CRC

characterized by having the ants move randomly for an arbitrary amount of time; haywire weighted causes the ants to follow the same pheromone that they are releasing. Both are described further below.

- *Pheromone rule*. Each generation an ant will release a quantity of pheromone into the cell that it is currently occupying. The pheromone rule performs this function.

The following items were considered to be part of the state of a cell:

- *Communication ID*. Gives a unique ID to each data source; primarily used to keep track of where an ant was spawned from.
- *Communication state*. Defines what type of cell this is (e.g. data source, data sink, free cell, etc).
- *Diffusion rate*. Determines how much pheromone will diffuse out of this particular cell and into neighboring cells each generation.
- *Evaporation rate*. Determines how much pheromone will evaporate out of this cell each generation.
- *Gossip probability*. The likelihood that this cell will take part in communicating its pheromone levels with its neighbors for a particular generation.
- *Will gossip*. Indicates if the current cell will communicate with its neighbors during this generation.
- *Occupied*. Denotes if there is an ant present in the current cell.
- *Return levels*. A list of all return pheromone levels separated by the originating data source for each ant.
- *Search levels*. A list of all search pheromone levels separated by the originating data source for each ant.
- *Source count*. The maximum number of data sources in the CA.

The following items were considered to be part of the state of an ant:

- *Haywire random*. Denotes that the ant is malfunctioning and is moving randomly.
- *Haywire weighted*. Indicates that the ant is malfunctioning and is following its own type of pheromone.
- *Lifetime*. Indicates how many generations an ant has been in existence.
- *Source address*. Indicates the location of the data source that spawned the ant.
- *Next address*. Denotes the ant's location.
- *Random*. This is the probability that an ant will move randomly for one generation.
- *Status*. Indicates whether the ant is searching or returning.

In Figure 46.3 we provide a sample execution of the ant pheromone algorithm. Walls are areas that do not allow transmission. The data source is located in the upper right corner, and is transmitting data to the data sink in the upper left corner. The search ants (return ants) are en route to the data sink (data source). The pheromone levels are represented by the darkness of the colored blocks that occupy most of the region; the darker blocks represent a higher concentration of pheromone.

46.3.6 Derivation of Parameters

The state of the ants and the cells in the ant pheromone algorithm and many of the rules that control the CA depend on numerous parameter settings, so that the model can be modified with relative ease to fit the application. Almost all of these parameters necessitate careful setup in order to get the algorithm to run at its peak performance. Determination of the proper parameter settings was calculated by varying one parameter at a time and executing the model for numerous runs.

To evaluate the parameters, two metrics have been utilized. Performance is the first metric, and is defined as the mean number of hops needed for a round trip journey from the nest to food for all ants in the simulation. The second metric is power, which is defined as the percentage of cells that change

© 2005 by Chapman & Hall/CRC

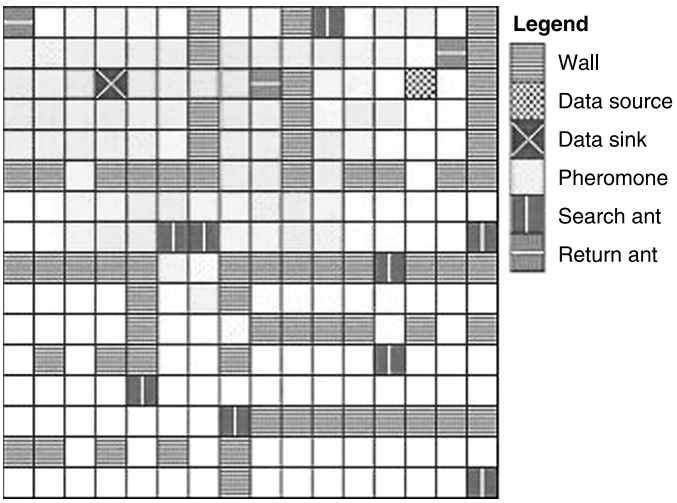


Figure 46.3. Example of ant pheromone algorithm designed with Cantor.

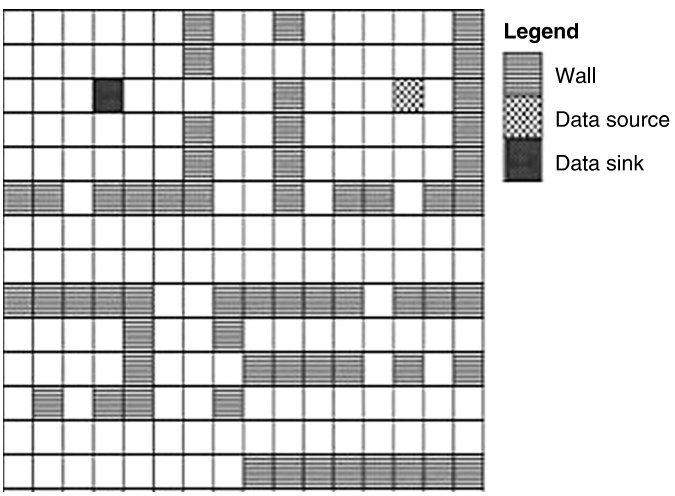


Figure 46.4. Floorplan used to derive system parameters.

their state each generation. Communication overhead is the dominant consumer of power in WSN applications, which is why this metric gives a reasonable measurement of system power.

Figure 46.4 shows the floorplan that was used to derive all the system parameters. Notice that the flow of data would originate from the data source, go to the data sink, and then terminate at the data sink. Each possible parameter value was executed for several runs, until the performance results fell within a 95% confidence interval of ten hops.

The following provides an explanation of what each of the critical parameters are, how they affect performance, and how they affect power.

46.3.6.1 Spawn Frequency

The rate at which a data source (ant nest) generates ants is controlled by a parameter setting that denotes the probability that a data source will spawn an ant in any particular generation. Figure 46.5 (on the left) shows how varying this parameter affects performance; increasing spawn frequency improves performance. However, only a small performance gain is possible from increasing this parameter beyond 50%. We have also evaluated how spawn frequency affects power, as is seen in

© 2005 by Chapman & Hall/CRC

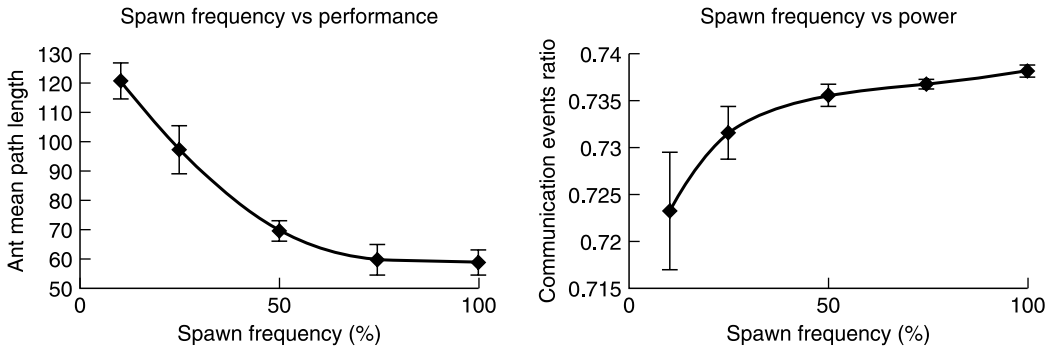


Figure 46.5. Effect of how rapidly ants are generated on performance (left) and power.

Figure 46.5 (on the right). The power profile increases quite rapidly for spawn frequencies below 25%. This is not surprising, as if there are very few ants laying down pheromone, then the cells' states will not change as rapidly. Beyond 25% the rate that power increases becomes quite small.

46.3.6.2 Repulsion Ratio

A pathology was noticed in our initial implementation, where ants that were moving to and from the data sink would cluster together and never reach their destinations. The reason this would occur is that if two ants that were searching and returning were to come into contact with each other then it would be likely that their respective pheromones would act as local maxima, and as a result the ants would be trapped. To counteract this, we caused the ants to be repulsed by the pheromone they currently emit. As a result, ants are pulled towards the pheromone that they are not releasing and they are pushed away from the pheromone that they are releasing. A parameter called the repulsion ratio was created denoting the relative strength of repulsion compared with attraction. This compels ants not to stay in one area and solved the pathology.

Figure 46.6 (left) plots performance versus the repulsion ratio. This parameter must be kept in the vicinity of 100% to achieve best performance (as this graph indicates), meaning that a setting above 100% will keep the ants from grouping together. However, as the parameter is increased beyond 100% the performance begins to deteriorate, since the ants begin to pay less attention to the ant trails. Figure 46.6 (right) shows how varying the repulsion ratio affects the power consumption of the algorithm. Analysis of this graph indicates that all parameter settings beyond 50% will have approximately the same power consumption. The low power exhibited by having a setting of 0% is a result of the ants clustering together and not moving, causing far fewer cell states to be affected by the ants, and as a result fewer messages are passed.

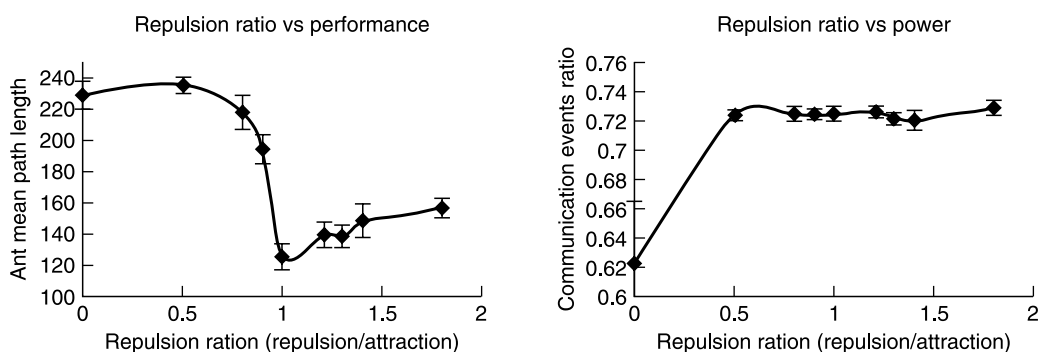


Figure 46.6. Effect of repulsion ratio on performance (left) and power (right).

© 2005 by Chapman & Hall/CRC

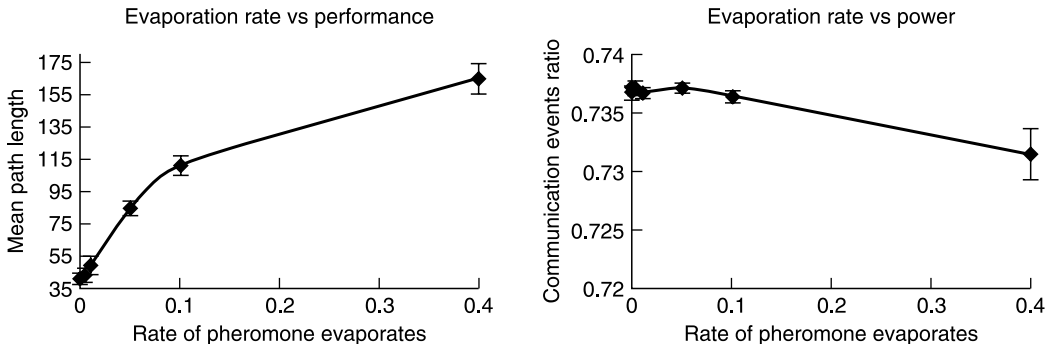


Figure 46.7. Effect repulsion ratio rate on performance (left) and power (right).

46.3.6.3 Evaporation

To keep the quantity of pheromone in the system from growing in an unbounded fashion we have included a parameter denoted as evaporation. In each generation of the algorithm a certain percentage of the ant pheromone evaporates (i.e. it disappears). As Figure 46.7 (left) indicates, any amount of evaporation tends to harm performance, as ant paths get erased before the ants can use them. Although it is not visible from this graph, parameter settings in the vicinity of 0.005 are optimal for performance. We have also analyzed evaporation based on power, as Figure 46.7 (right) indicates. Not surprisingly, increasing the value of this parameter helps to reduce the power of this algorithm. Unfortunately, any parameter setting that would result in significant power savings coincides with greatly decreased performance.

46.3.6.4 Random Ant Movement

Utilizing a random component in a computer algorithm is oftentimes beneficial towards increasing performance. Randomness was included in this algorithm by giving the ants a probability that they will move randomly for one generation rather than follow the pheromone gradient. The performance impact of this randomness is presented in Figure 46.8 (left). We have determined that a parameter value below 15% increases performance, whereas beyond 15% it starts to reduce performance. The optimal setting is around 5%. We have also evaluated the random movement's effect upon power consumption. The results are provided in Figure 46.8 (right). This graph shows that having the ants move randomly is beneficial to reducing the power consumption of the algorithm.

46.3.6.5 Diffusion

As stated previously, the ant pheromone spreads to other cells; the amount of pheromone that diffuses out of the cell and into its neighbors is controlled by the diffusion rate parameter. This parameter's

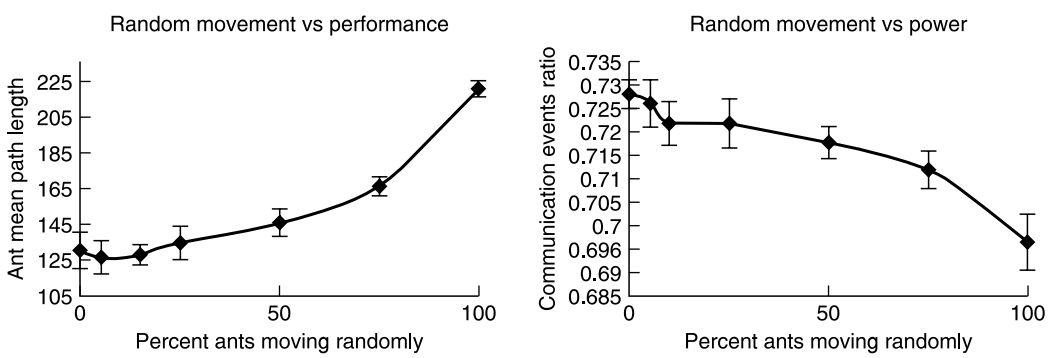


Figure 46.8. Effect of random ant movement on performance (left) and power (right).

© 2005 by Chapman & Hall/CRC

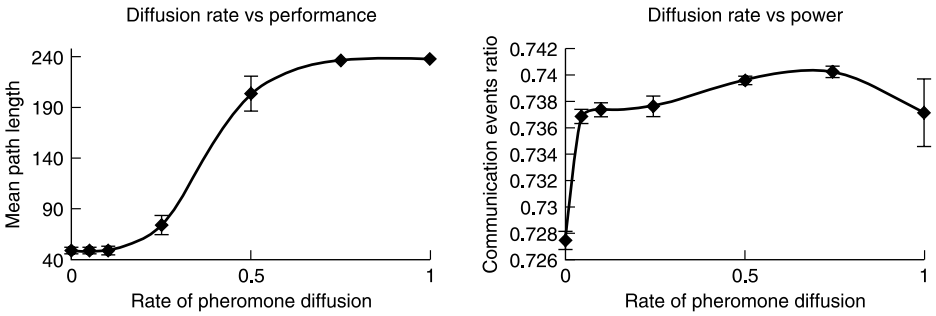


Figure 46.9. Effect of pheromone diffusion on performance (left) and power (right).

effect upon performance is provided in Figure 46.9 (left). A setting of 0.05 appears to works best. A setting beyond 0.2 begins to have a large negative impact on performance. The effect of diffusion upon power is illustrated in Figure 46.9 (right). This graph indicates that a low diffusion rate coincides with good performance and low power. The low diffusion setting exhibits low power since the pheromone does not move as quickly, while extremely high settings have low power since the pheromone moves so fast that a steady-state behavior occurs quite rapidly.

46.3.7 Exploration of Errors

To determine how robust ant pheromone is to errors, the algorithm was subjected to a number of different types of error condition. These error conditions are: (i) haywire random (a random selection of ants will move randomly for a random amount of time); (ii) haywire weighted (a random selection of ants will follow the opposite pheromone); (iii) haywire cells (a random selection of cells will produce a random amount of pheromone). How these error conditions affect performance is illustrated in Figure 46.10 (left). Affecting ants with up to 50% of haywire random increases performance. The effects of haywire cells and haywire weighted are similar, in that they reduce performance rather drastically up to about 25%, where further hits to performance begin to level off. In Figure 46.10 (right) we show how these error conditions affect the power consumption of the algorithm. The power consumption for haywire cells is quite dramatic: anything beyond 25% is near maximum power consumption. The decrease in power consumption attributed to haywire weighted can be attributed to how this behavior tends to cause the ants to conglomerate into groups and not move, which has the side effect of reducing communication.

46.3.7.1 Conclusions on Errors

From the preceding analysis it has been concluded that ant pheromone allows for routing that is very robust to a number of errors, even when much of the network is being afflicted with various pathologies.

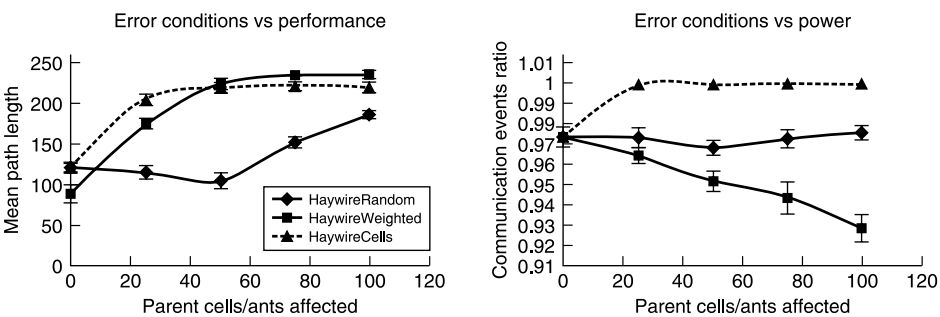


Figure 46.10. Effect of error conditions on performance (left) and power (right).

© 2005 by Chapman & Hall/CRC

46.3.8 Why Pheromone Works

The following presents an informal explanation of why pheromone routing works. A CA that shows how pheromone evaporates and an informal proof have been provided.

46.3.8.1 Pheromone Simulation

A simple CA was developed to show the behavior exhibited by the pheromone utilized in an ant pheromone CA. This CA allows for an arbitrary collection of ant pheromone to be placed into a lattice, and with time it will evaporate and diffuse throughout the region.

Figure 46.3 shows one particular execution of this CA. In this example, a curve of ant pheromone is laid down at time zero, as is indicated by the black shape in Figure 46.11(a). The intersection of the lines is provided to denote the center of the curve in each of the figures. The concentration of the pheromone is indicated by its relative darkness in the figure, where black is the highest concentration. Figure 46.11(b) shows the pheromone during generation ten; notice how the regions of highest pheromone concentration are below the original curve. As time progresses, the region of highest pheromone concentration begins to align itself with the straight line that connects the endpoints of the curve (i.e. the optimal path). As even more time passes, the path begins to fade into background noise.

Now consider what would happen if other ants were to approach this region. The following ants will mainly follow the highest concentration of pheromone. Shortly after the initial pheromone was laid down, the highest concentration is a slightly flatter curve that is closer to the optimal path than the initial curve. As more time passes, the area of highest concentration will flatten further, until the pheromone gradient is essentially a circle with the pheromone level increasing as it gets towards the center. As a long time passes, the pheromone levels in a local region will tend to be the same and the pheromone will act as background noise.

The intended point of this example is that ants following another ant's trail will tend to follow a more optimal path with time. If one considers the path of an ant to be composed of several consecutive

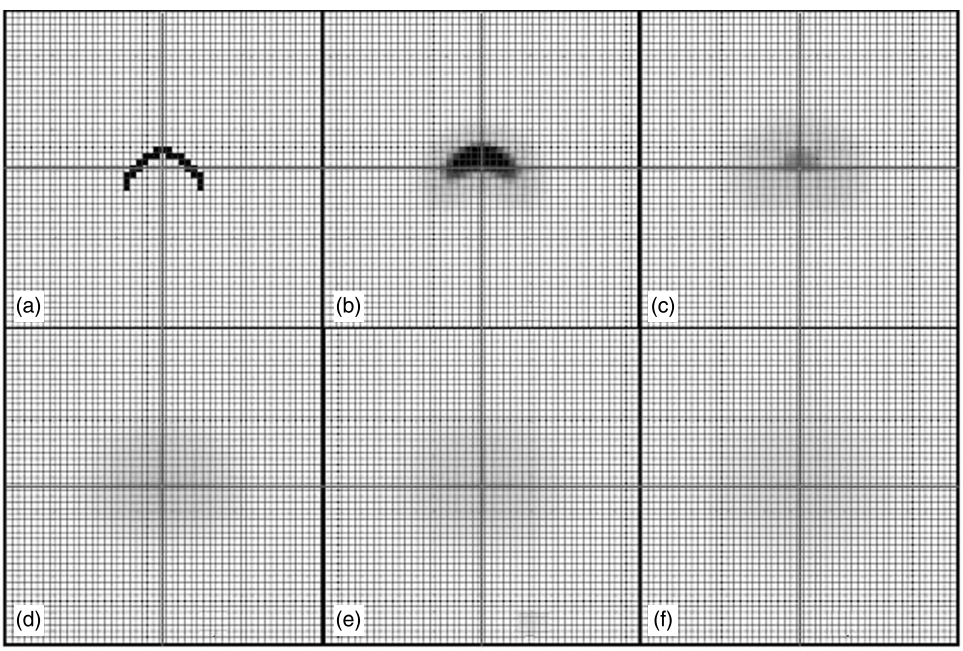


Figure 46.11. Pheromone diffusion at work. (a) An initial curve of pheromone is laid down at time zero. As time progresses, the pheromone diffuses as shown by (b) at time 10, (c) at time 20, (d) at time 40, (e) at time 60, (f) at time 100.

© 2005 by Chapman & Hall/CRC

curves, and that with time the pheromone will be at its highest concentration along a straight line connecting the beginning and end of the curve, then the net result will be an increasingly optimal route that other ants can follow.

The second point to be taken is that transient paths that do not lead to the ants destination will eventually be lost as background noise.

46.3.8.2 Pseudo-Proof

What follows is an informal proof of the concept illustrated in the previous simulation, namely that ant pheromone will convert curved paths into optimal straight lines.

As stated in the last example, the path of an ant can be considered to be a sequence of consecutive curved paths, where a curve is defined as being two consecutive inflection points along an arbitrary continuous function, and the optimal route would be defined as the straight line connecting the two inflection points.

To help understand the behavior of ant pheromone, an iterative equation was developed:

$$P_{x,y,t} = (1-E)P_{x,y,t-1}(1-\rho F) + L(x,y,t) + \sum_{i=1}^{\sqrt{N+1}} \sum_{j=1}^{\sqrt{N+1}} (1-E)P_{i,j,t-1}\frac{F}{N}\rho s(i,j) \tag{46.2}$$

where $P_{x,y,t}$ is the pheromone level at location (x, y) at time t, E is the evaporation rate in $[0, 1]$, ρ is the gossip probability in $[0, 1]$, F is the diffusion rate in $[0, 1]$, $L(x,y,t)$ is the path of ant for time t, N is the size of the neighborhood, and

$$s(i.i)\begin{cases} 0 & \text{if } i = j \\ 1 \end{cases}$$

To simplify matters, the following assumptions were made:

1. An instantaneously formed path of pheromone (e.g. the path in Figure 46.11) approximates the behavior of having an ant follow the same path laying down pheromone in each generation.
2. Perpendicular cross-sections along the ant path are independent of each other. Specifically, it is assumed that the pheromones in one cross-sectional region do not have an effect in another cross-sectional area.
3. The evaporation rate should not be allowed to be exceedingly large.

Justification of the assumptions now follows:

1. This assumption can be explained best in two separate parts:
   a. Consider sections of the ant path that are close to each other. These sections will usually have been laid down within a short time interval of each other. And since there is only a small disparity in time for these local sections of the ant path the behavior of the entire ant path can be approximated by sections that are laid down instantaneously across local regions.
   b. Now consider sections of the ant path that are further apart from each other. These sections of the ant path were laid down in a relatively large time interval from each other. However, as the portions of the ant path are further apart, their respective pheromones will have negligible effect upon each other, as is indicated by looking at Equation (46.2). This equation indicates that the further the pheromone travels the more it gets diffused. Consequently, it should be clear that their pheromone interactions will get progressively weaker as two portions of the ant path get further apart. As such, it can be assumed that the entire ant path can be approximated as instantaneously laid down.

© 2005 by Chapman & Hall/CRC

2. Consider the entire ant path as a conglomeration of sections that are perpendicular to the ant path. The true behavior of the ant pheromone is that each cross-section will also affect its neighboring cross-sectional areas. Now consider the pheromone behavior of cross-sectional areas that are relatively close to each other. Since these sections are close to each other they will be affected primarily by the same sections of the ant path, and hence their pheromone behaviors will be similar. Now consider the behavior of diffusion across neighboring cross-sectional areas. A particular cross-section will diffuse equivalent quantities of pheromone into its neighbors. Likewise, its neighbors pass pheromone back into it. If a cross-sectional area's pheromone is similar to those of its neighbors, then it can be assumed that the pheromone leaving the region is similar to the pheromone coming in. Thus, the cross-sectional areas can be considered to be independent of each other.
3. For the second assumption to be valid the evaporation rate cannot be too high. Since this is entirely controlled by a parameter setting, this can be readily done. Fortunately, a high evaporation rate is not particularly interesting, since most ant paths are reduced to nothing quite rapidly, making pheromone routing largely ineffective.

Given Equation (46.2), if one were to look at any point along the ant path during the instant that the ant laid down the pheromone then the following behavior would be seen:

1. Initially, the pheromone would all be located along the ant trail.
2. As time progresses, Equation (46.2) dictates that the pheromone will spread. Specifically, the pheromone will move one hop further from the initial ant path each generation. It can be seen from Equation (46.2) that, in each cross-sectional area, the pheromone will resemble a curve. The peak of the curve will always be located along the ant path. The further from the ant path a particular point is, the less its pheromone level will be.
3. As time progresses, the curve of pheromone in a cross-section will expand to new regions. The curve will also tend to flatten over time. This can be seen by examining Equation (46.2), and seeing that large local differences in pheromone will smooth out with time.

The ant's path divides the region into two portions. On one portion the cross-sectional areas will point towards the center of the curve (call these sections type A), while in the other portion the cross-sectional areas will point away from the center of the curve (call these sections type B). Given the symmetry of the pheromone model, both of these regions will have the same amount of pheromone within them.

The type B cross-sectional region will diffuse into a larger area than the type A cross-sectional region. Therefore, the individual pheromone levels in the type A cross-sectional region will be higher than the pheromone levels in the type B cross-sectional region. This will cause the ants to want to follow the pheromone in the type A region, which contains the optimal path (i.e. the straight line connecting the curve's endpoints).

46.3.9 Gossip

46.3.9.1 What Is Gossip?

Gossip is a communication method that can be used to replace message flooding. Consider the following example using flooding. A node in a sensor network wants to send a message to all other nodes in the network, so the node then broadcasts the message to all of its neighbors. These neighbors, in turn, broadcast the message to all of their neighbors. Each time a node receives such a message for the first time it will broadcast it. When a node receives a duplicate message it will not broadcast it. Eventually the entire network will have received the message, but at the expense of huge quantities of messages. With gossip, instead of always routing a received message to all of its neighbors, a node will probabilistically pass on the message; that is to say, there is a probability that the node will broadcast the message. It has been shown that this technique greatly reduces the amount of messages required to get

© 2005 by Chapman & Hall/CRC

a message to all the nodes in a region, and with proper parameter settings this technique can get a message sent to all nodes with a near-unit probability with far fewer messages than would have been required with flooding. An excellent example of a network that utilizes several different types of gossip and shows the effect of varying key parameter values is provided by Haas *et al.* [5]. For more in-depth theory on gossip, see Kempe *et al.* [6], Chandra *et al.* [7], and Wokoma *et al.* [8].

46.3.9.2 How It Was Used?

The probabilistic message passing idea from gossip has been applied to how pheromone behaves in the ant pheromone algorithm. Diffusion of pheromone in particular was modified. Now, a node will have a certain probability that it will diffuse its pheromone upon its neighbors. By examining Equation (46.2), one should be able to see that including gossip in this manner will slow down the diffusion of pheromone. It will also make the overall pheromone gradient much more sporadic the less frequently that pheromone information is shared.

To illustrate the effects of utilizing gossip, two separate examples, similar to Figure 46.11, have been provided. Again, an initial curve-shaped path of pheromone is laid down at time zero, as shown in Figure 46.12(a) and Figure 46.13(a), and as time progresses the pheromone diffuses throughout the region. In the example from Figure 46.11, every generation a cell would diffuse some of its pheromone to its neighbors. In Figure 46.12 there is a 0.25 probability that a cell will diffuse its pheromone, and in Figure 46.13 there is a 0.75 probability that a cell will diffuse its pheromone.

As these examples indicate, utilizing gossip in the pheromone tends to make the pheromone gradient much less smooth. Further, the less probabilistic a cell is to pass its pheromone information, the longer it takes for the pheromone region to expand.

46.4 Summary

Biological primitives can be a very important tool for studying and designing applications for WSNs. Much work has been done in the discipline of biology to model the behavior of naturally occurring self-organizing systems. A systematic way of modeling such complicated systems has been adapted to WSNs and is provided here. This chapter provides an in-depth case study of one particular biological primitive, ant pheromone, which was used to develop a routing algorithm in a WSN for a MOUT

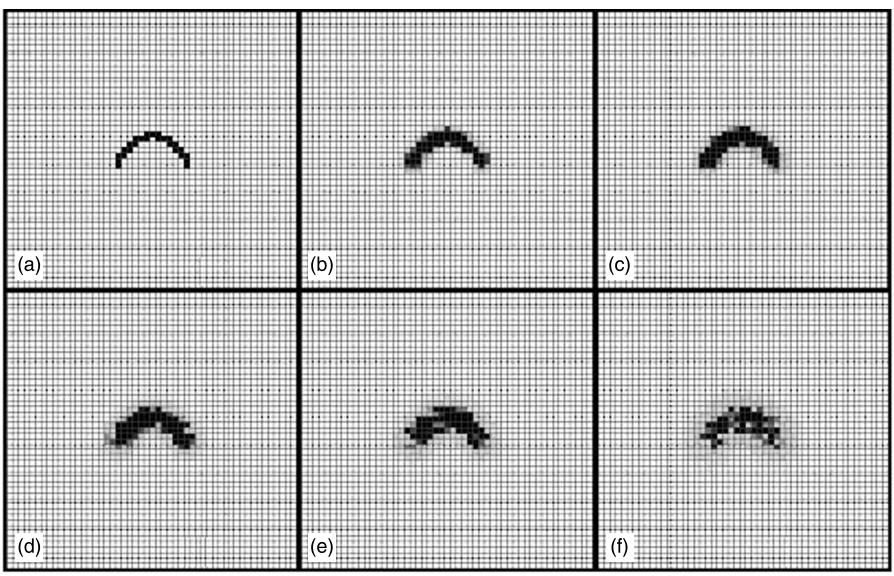


Figure 46.12. The effect of a gossip probability of 0.25. (a) An initial curve of pheromone is laid down at time zero. As time progresses, the pheromone diffuses as shown by (b) at time 10, (c) at time 20, (d) at time 40, (e) at time 60, (f) at time 100.

© 2005 by Chapman & Hall/CRC

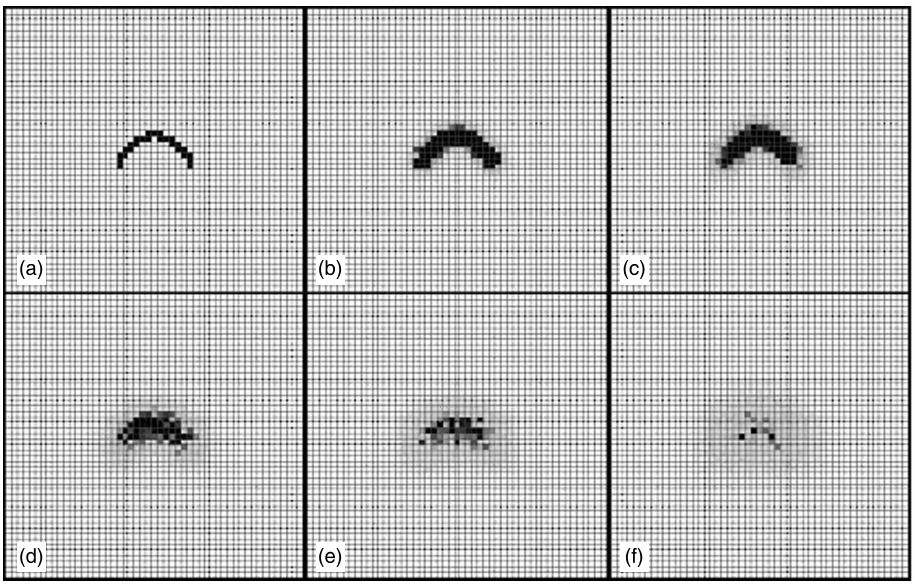


Figure 46.13. The effect of a gossip probability of 0.75. (a) An initial curve of pheromone is laid down at time zero. As time progresses, the pheromone diffuses as shown by (b) at time 10, (c) at time 20, (d) at time 40, (e) at time 60, (f) at time 100.

application. Explanation of the tools used, parameter settings, optimizations, performance, and power consumption were provided.

Acknowledgment and Disclaimer

This chapter is partially supported by the Office of Naval Research under Award No. N00014-01-1-0859 and by the Defense Advanced Research Projects Agency (DARPA) under ESP MURI Award No. DAAD19-01-1-0504 administered by the Army Research Office. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author and do not necessarily reflect the views of the Office of Naval Research (ONR), Defense Advanced Research Projects Agency (DARPA), and Army Research Office (ARO).

References

[1] Camazine, S. *et al.*, *Self-Organization in Biological Systems*, Princeton University Press, Princeton, NJ, 2001.

[2] Chang, K. *et al.*, Joint probabilistic data association in distributed sensor networks, *IEEE Transactions on Automatic Control*, AC-31, 889, 1986.

[3] Dorigo, M. *et al.*, The ant system: optimization by a colony of cooperating agents, *IEEE Transactions on Systems, Man, and Cybernetics Part B*, 26(1), 29, 1996.

[4] Di Caro, G. and Dorigo, M., AntNet: a mobile agents approach to adaptive routing, Technical Report IRIDIA/9712, Université Libre de Bruxelles, Belgium, 1997.

[5] Haas, Z. *et al.*, Gossip-based *ad hoc* routing, in *Proceedings of the IEEE INFOCOM*, 2002.

[6] Kempe, D. *et al.*, Spatial gossip and resource location protocols, in *Proceedings of 33rd Annual ACM Symposium on Theory of Computing*, 2001, 163.

[7] Chandra, R., *et al.*, Anonymous gossip: improving multicast reliability in mobile *ad-hoc* networks, in *Proceedings of 21st International Conference on Distributed Computing Systems*, 2001, 275.

[8] Wokoma, I. *et al.*, A weakly coupled adaptive gossip protocol for application level active networks, in *IEEE 3rd International Workshop on Policies for Distributed Systems and Networks*, 2002.

© 2005 by Chapman & Hall/CRC

47

Physics and Chemistry

Mengxia Zhu, Richard Brooks, Matthew Pirretti, and S.S. Iyengar

47.1 Introduction

47.1.1 Wireless Sensor Networks

A wireless sensor network (WSN) is a set of wireless sensor nodes that sense, process data, and communicate cooperatively. Sensor node location and network topology are typically not predetermined. Models for node deployment should thus be at least partially stochastic. WSNs have both military and civilian applications. WSN system designs need to satisfy a number of constraints simultaneously:

- WSN nodes typically rely on battery power, making eventual node failure inevitable. Whereas wired network protocols are usually designed to provide a high quality of service (QoS), energy conservation is often a major issue for WSN implementations. There is a trade-off between QoS and system lifetime in these systems.
- WSNs use low-cost radios with a high error rate and limited bandwidth. Associated protocols need to have low communications overhead and high fault tolerance. Limited on-board processors and memory prohibit the use of overly complicated protocols.
- Sensor node failures and possible node mobility make network topology transient. The environment also has frequent and unpredictable perturbations. These topological disturbances demand self-organizing protocols, capable of adapting to these changes.
- WSNs must also have distributed control architectures to maintain a level of reliability, scalability, and flexibility that is not possible for centralized control systems.

47.1.2 Routing in WSNs

Routing protocols determine the paths that sensor nodes use to communicate. *Ad hoc* routing protocols have two main variants: (i) table-driven and (ii) demand-driven.

Table-driven protocols maintain routing tables describing paths between all nodes in the network. Topological changes propagate throughout the network. Routing tables need to be modified on each

© 2005 by Chapman & Hall/CRC

node. The storage requirement of the routing tables and the transmission overhead of topological changes are the largest drawbacks of this approach. An example table-driven routing protocol is destination-sequenced distance vector routing (DSDVR).

Demand-driven routing protocols run at the request of a source node. The control overhead is much less than for table-driven protocols. This approach has a route discovery phase, followed by a route maintenance phase, until either the destination becomes inaccessible or the route is no longer needed. An example protocol is *ad hoc* on-demand distance vector (AODV) routing.

There is a vast range of complex physical, chemical, biological, and social systems in the real world. These systems consist of many interacting individuals following simple behaviors. Although individual behaviors appear simple, the collective behavior emerging from interaction among numerous individuals can be very complex. For instance, convection in fluid dynamics, particle diffusion in chemistry, and food foraging in social insects are all examples where individual components have simple behaviors, yet the group has a complex global behavior. These natural mechanisms compel computer scientists to design distributed algorithms, which can be emulated using simple computing devices. The cellular automaton, first conceived by John von Neumann in the early 1950s, serves as a tool for modeling collective behaviors. This chapter shows how the Ising model from physics and diffusion-limited aggregation (DLA) from chemistry can help solve the routing problem in WSNs.

47.2 Discussion of Two Examples from Physics and Chemistry

At first glance, there may appear to be little correlation between the Ising model and DLA and routing in a WSN. However, modifications (provided in Section 47.5 of this paper) to the two systems make them excellent models for solving the WSN routing problem.

47.2.1 Ising Model

The name of this model derives from the German physicist Ernst Ising, who first introduced a simple mathematical model of magnetism in 1925. This model serves as one of the most important models of statistical physics. The Ising model is composed of atomic magnets that can be viewed as magnetic vectors pointing either to the north or the south poles. Suppose we have N such little magnetic spins s_i, $i = 1, 2, 3, \ldots, N$ on a two-dimensional lattice with each pointing up ($s_i = +1$) or down ($s_i = -1$). Each spin interacts with its nearest neighbors in addition to an overall external magnetic field.

There are 2^N different configurations for N magnetic spins. The quantum Hamiltonian energy for a configuration of spins $\{s_i\}$ is given by

$$E[\{s_i\}] = -H \sum_{1}^{N} s_i - K \sum s_i s_j \tag{47.1}$$

Magnetization can be computed as

$$M = \sum_{1}^{N} s_i \tag{47.2}$$

The first term in Equation (47.1) represents the coupling energy between each spin and external field. H is the coupling constant of the external field. The second term explains the interaction energy between all neighboring spins. K defines the strength of spin–spin interaction; K is positive for a ferromagnetic bond (i.e. spins with parallel directions have lower energy) and negative for an anti-ferromagnetic bond (i.e. spins with anti-parallel directions have lower energy). Here, we assume periodic boundaries (the right neighbor of a spin on the right edge is the corresponding spin on the left edge) and no external

© 2005 by Chapman & Hall/CRC

magnetic field. To find the ground state of the system, a brute-force method would require considering all $2N$ configurations, as is done for the traveling salesman problem by testing all possible routes among all N cities. Clearly, the Ising model is in the class of NP-complete problems.

From Equation (47.2), we see that we can get a fairly large magnetization if most spins adopt the same direction, as shown by Figure 47.1 (right); otherwise, random spin arrangement with opposite directions counteract each other and results in an imperceptible magnetization as shown by Figure 47.1 (left).

The spin-glass variation of the Ising model contains both ferromagnetic and anti-ferromagnetic bonds. In most cases, spins point in random directions and no macroscopic magnetic field is formed because of cancellation among the individual magnets. However, in some metals, like iron, numerous magnetic vectors align to produce a perceptible macroscopic magnetic field. The phase transition between magnetization and nonmagnetization is tuned by a thermodynamic variable, temperature (similar to how water turns into ice when it is cooled). Energy minimization dominates under low-temperature conditions, resulting in a macroscopic magnetization where most of the spins are aligned. Entropy maximization dominates system behavior when the temperature is high; there is no net magnetization, since spins point in random directions. Figure 47.2 shows an approximation of net magnetization as a function of temperature. Magnetization is high at low temperatures and decreases as



Figure 47.1. Left: random spin direction arrangement. Right: spin direction align (applet from http://bartok.ucsc.edu/peter/java/ising/keep/ising.html).

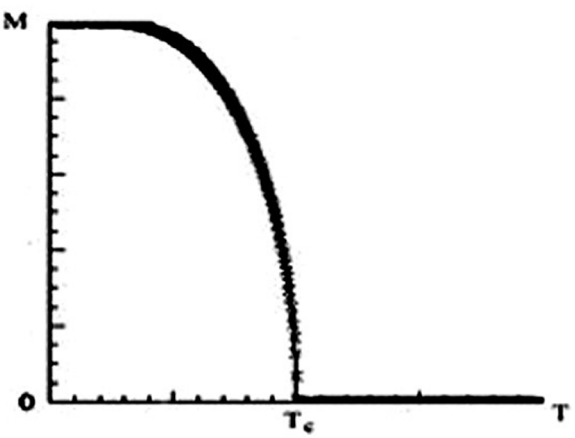


Figure 47.2. Magnetization versus temperature [1].

© 2005 by Chapman & Hall/CRC

temperature increases. A sudden plunge is noticed when temperature exceeds critical temperature, T_c and is a type of phase transition.

The probability for each possible microscopic spin configuration is defined by the Boltzmann distribution function:

$$P[\{s_i\}] = \mathrm{e}^{-E(\{s_i\})/KT} \tag{47.3}$$

where $E(\{s_i\})$ is the energy of the system in state $\{s_i\}$, K is Boltzmann's constant, and T is the temperature in degrees kelvin.

However, in order to make probabilities of all configurations add up to one, a normalization factor is calculated in the following partition function:

$$Z = \sum_A \mathrm{e}^{-E(A)/KT} \tag{47.4}$$

where $E(A)$ is the energy of the system in all possible configurations. Equation (47.5) gives the actual probability of the system staying at configuration $\{s_i\}$:

$$P'[\{s_i\}] = \frac{\mathrm{e}^{-E(\{s_i\})/KT}}{Z} \tag{47.5}$$

47.2.2 Fractals

Highly regular geometries have been studied intensively by mathematicians throughout history, but it was not until the late 1970s that nested shapes with arbitrarily intricate pieces began to attract more attention [2]. Benoit Mandelbrot first proposed the notion of a fractal, which is an irregular geometric object with an infinite nesting of structure at all scales. Fractals are commonly observed in nature, such as coastlines, clouds, rivers, and trees. Fractals refer to those objects that have fractional dimensions as opposed to integral dimensions in Euclidean and Cartesian geometries. The fractional dimension serves as a quantified index describing the complexity or texture of the object's surface. Another property of a fractal is self-similarity: it looks similar at all levels of magnification. For example, the fern leaf in Figure 47.3 looks similar whether viewed at life size or in detail at high magnification.

Euclidean space tells us that a line has one dimension, a plane has two dimensions, and a cube has three dimensions. Another dimension definition is based on the number of variables in a dynamic



Figure 47.3. Fern leaf.

© 2005 by Chapman & Hall/CRC

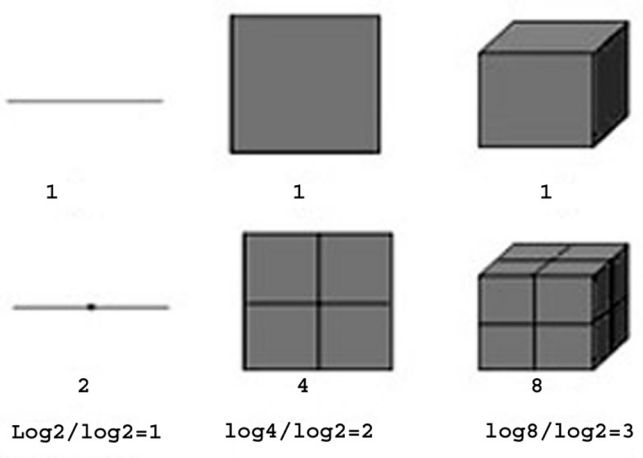


Figure 47.4. Example of Hausdorff dimension calculation when scaling factor $s = 2$.

system. The Hausdorff dimension, which was first introduced by Felix Hausdorff in 1919, defines an accurate way of measuring the fractal dimension. The Hausdorff dimension agrees with the Euclidean dimension on regular shaped objects. Let us look at a simple example in Figure 47.4. We divide each spatial linear size equally by s. Equation (47.6) computes the dimension for each case.

$$D = \frac{\log(N)}{\log(s)} \tag{47.6}$$

where N is the number of similar objects and s is the scaling factor.

We show another example of how to calculate the numerical value for the fractal dimension of a coastline. Let us try to obtain the perimeter of the coastline in Figure 47.5. Suppose we first use a ten-unit ruler to measure it: we get 11 segments. If we use a smaller, five-unit ruler, then we get 23 segments, which means that we get a larger perimeter with a smaller ruler. As the size of the ruler is progressively scaled down we tend to get a perimeter that approaches an infinite value. The more irregular the coastline appears, the easier it is to observe this property.

The fractal dimension of an object is specified as being one plus the absolute value of the ratio of the logarithm of the increase in perimeter to the logarithm of the increase in scale. The fractional dimension

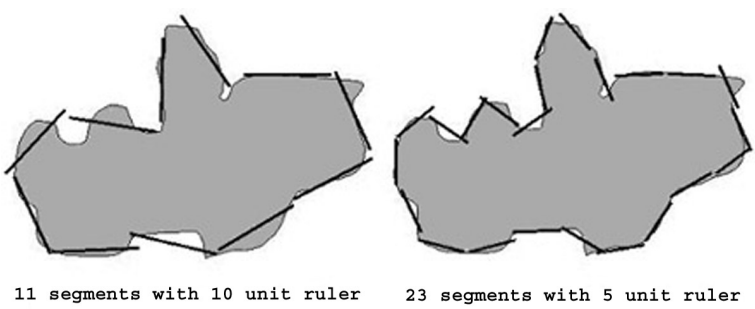


Figure 47.5. Coastline perimeter measurements.

© 2005 by Chapman & Hall/CRC

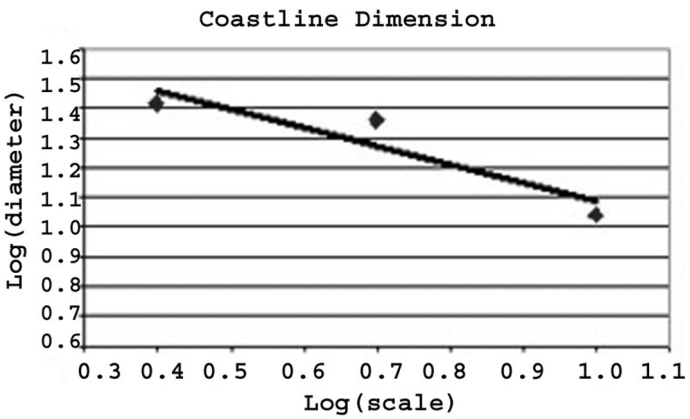


Figure 47.6. Log–log graph of coastline derived from an object's perimeter and the scale of the ruler that measured it.

allows us to correlate these values with physical properties. Fractal dimensions can be related to physical processes, as in the following examples:

- Rocks: related to their erosion.
- Fractures in oil-bearing deposits: related to the most economical rate of extraction.
- A topographic representation of disease spread: related to the agent's virulence.
- The outline of a forest fire: related to its speed of spread and the difficulty of extinguishing it [3].

Observe Figure 47.6, where the scale of the ruler versus the measured perimeter is plotted on a log–log graph: a best-fit line can be drawn through the points. The slope of this regression line is −0.623. Based on this, the fractal dimension is specified as being 1.623. If the coastline appears more irregular, then the fractional dimension would be larger.

Studies show that there are two separate processes that create fractals: multiplicative iteration of random processes, which create a multi-fractal structure, and additive processes, which generate mono-fractals [3]. A single exponent can define a mono-fractal, whereas hierarchies of exponents are required to depict multi-fractals. The noninteger exponent index is called the fractal dimension.

47.3 Modeling Tool

47.3.1 Cellular Automata Background

Cellular automata (CAs) are composed of an array of identically programmed automata, or ''cells'', which simultaneously interact with neighboring cells and advance in discrete time and space in any arbitrary number of dimensions. Each cell has its own states and rules to compute what its next new state shall be, based on its own state and its neighbors' states. There is no centralized authority to control the global behavior; control is entirely through the local interaction among identical entities. This unique property of CAs makes them an invaluable tool to model self-organizing systems.

Although the local rules of the individual entities in most self-organized systems are well documented, there is still uncertainty as to the exact nature of the global behaviors, as knowledge of local behaviors does not imply that global behaviors will also be straightforward. Generalized probabilistic rules are proposed to specify the likelihood of each particular state, where each cell evolves into one particular state out of several possible states in a probabilistic fashion. This turns out to be a very powerful extension to traditional deterministic rules. More natural phenomena can be modeled with this extension included. Our two routing models take advantage of this nondeterministic property, as can be seen in Section 47.5.

© 2005 by Chapman & Hall/CRC

Although CAs are a natural way to model discrete dynamic and complex phenomena, this flexibility and generality comes with a cost: instead of several variables used in a partial differential equation to describe a continuous dynamical system, a large number of variables are needed for a CA system. This is because a large number of cells are required for a large-scale system with complex behaviors. Moreover, the number of time steps required for evolution is considerably large [4].

47.3.2 Cantor Tool

The Cantor tool is a CA simulator based on the generic automata with interacting agents (GAIA) model. The inclusion of agents allows for cells that can move throughout the simulation space, interacting with other agents and cells alike. Cantor is capable of modeling dynamic, discrete time, and discrete space event systems, which consist of large interacting individuals that display collective phenomena. Cantor was developed by the Applied Research Laboratory at the Pennsylvania State University in 2001 and has been successfully used to simulate many CAs systems, including traffic engineering, network communication, and sensor data routing, etc.

47.4 Idealized Simulation Scenario

Figure 47.7 shows an idealized military operations in urban terrain (MOUT) scenario. Walls signify any obstruction that can block radio signals. Open cells are open regions allowing signal transmission. Open doors (closed doors) are choke points for signals that periodically allow (disallow) transmission. Finally, obstructions are intermittent disturbances that occur at random throughout the sensor field. Random factors are inserted to emulate common disruptions for this genre of network. Each square capable of transmission contains a sensor node. This amounts to having a sensor field with a uniform density. This provides an abstract example scenario approximating situations likely to exist in a real MOUT situation.

In Figure 47.7, the directions in which routing takes place are denoted; the neighboring cells being pointed to are the next hop on the way to the data sink. Roughly speaking, routes are chosen according to four different metrics:

- Maximum available power route: the route has the maximum amount of total available power.
- Minimum energy route: the route that consumes the minimal sum of transmission power.
- Minimum hop route: the route with the minimal number of hops to the data sink.
- Maximum minimum available power node route: the route along which the node with the minimum available power is larger than nodes with the minimum available power along other routes [5].

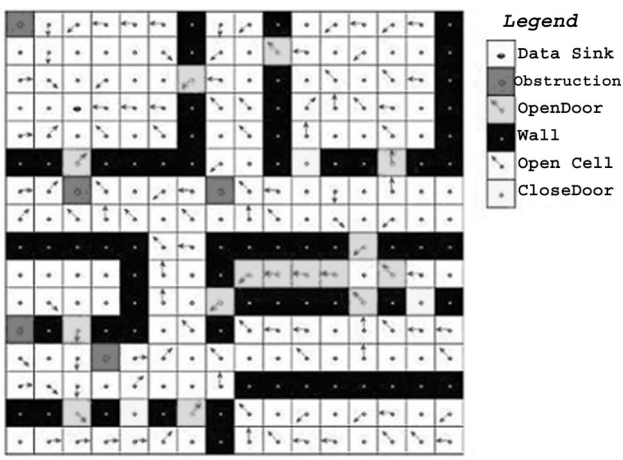


Figure 47.7. Example of a MOUT application.

© 2005 by Chapman & Hall/CRC

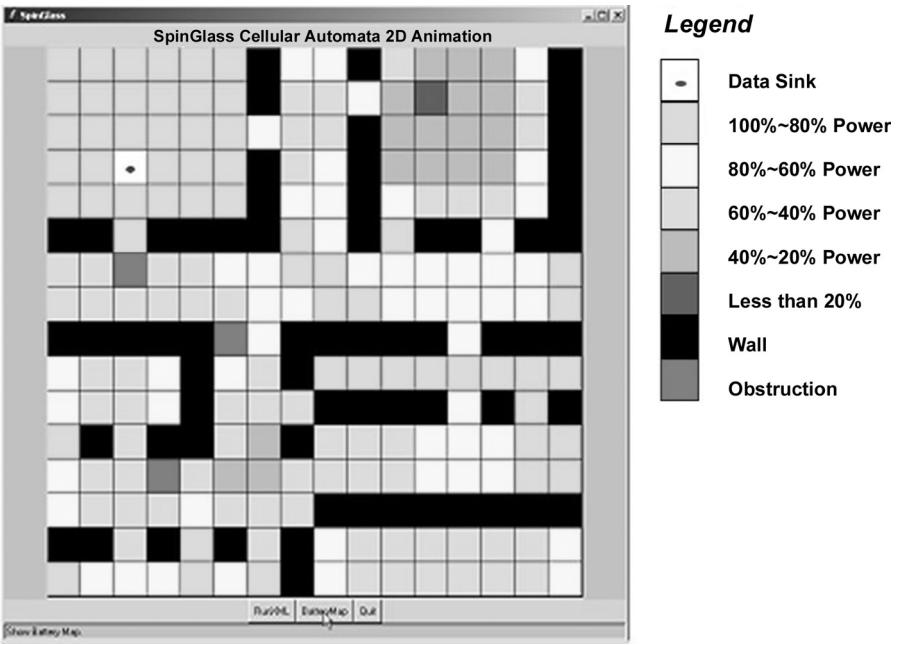


Figure 47.8. Energy map for WSN. The darker the color is, the less battery power remaining.

The minimum energy route scheme and minimum hop route scheme come up with the same results, provided that the same amount of transmission energy is required for every link.

As we discussed earlier, energy conservation is a crucial issues for WSNs. It would be very helpful if the end user could be aware of the remaining energy at each sensor node. Proper actions could be taken based on an energy map. For example, low-activity sensor nodes could be put into sleep node to save energy. Supplemental deployment could be launched to replace sensor nodes with a low-battery condition. Cantor can show the energy map for each discrete time step in a particular simulation. Figure 47.8 shows an example of an energy map. Every node starts with full energy. Communication amongst the sensor nodes for route establishment and adaptation to topological disturbances are considered to be the only consumers of energy. The colors of the squares are used as an indicator for the remaining battery at each sensor node, namely the darker the square is, the less remaining battery associated with that sensor node.

47.5 Applying Physics and Chemistry to WSN Routing

47.5.1 Spin-Glass Model

47.5.1.1 Adapt Spin Glass Model to WSN Routing

A simplified spin-glass model, which retains the essence of the Ising model, yet is simple enough, to be solved and simulated by a computer, was derived from real-world iron magnetism. In fact, many breakthroughs in science have been based on the study of some ''toy models'' emulating a much more sophisticated real system. Though the correlation between the spin-glass model and routing in WSNs may not be obvious, proper modifications to the spin-glass model make it very relevant to WSN routing.

In contrast to normal up and down spin directions, our sensor nodes can point to the eight different cardinal directions. This spin configuration belongs to the Potts spins type, which allows each spin to have p different values $0, 1, \ldots, p-1$. It is different from classical physics, in which a magnet can point to any direction it pleases. The nonzero energy difference associated with all directions is critical for the large-scale system correlation [6].

© 2005 by Chapman & Hall/CRC

In WSNs, hundreds or thousands of sensor nodes are deployed to monitor a particular region of interest. Data collected by the sensor nodes is typically sent to data sinks for postprocessing. Thus, traditional peer-to-peer routing philosophies, which establish a route between a source and destination address pair, are less pertinent. A routing mechanism capable of establishing data routes from sensor nodes to data sinks is needed.

Owing to radio transmission limitations, data are relayed to the sink in a multi-hop manner. Multi-hop communication consumes less power than single-hop communications by keeping transmission power levels low. In our spin-glass routing model, a dynamic potential field defining the minimum transmission energy to reach a data sink is initially established through local interactions of the sensor nodes. Then, a potential field together with a kinetic factor define spin directions of each cell by following the Boltzmann distribution function in Equation (47.5). Nevertheless, the formidable number of all possible configurations required in the denominator of the function prevents us from following the Boltzmann distribution function strictly. A brute-force method would require up to 8^N possible configurations of a grid with N cells. Instead, by only using local information, eight local configurations are calculated in the denominator in our model. This not only reduces computation load dramatically, but it also excludes global information. We use $T[n_i]$ to represent the potential energy value of node n_i. Node n_j, as one of eight neighbors of node n_i, has potential energy value denoted by $T[n_j]$. The probability $P[n_{ij}]$ that node n_i points to neighbor n_j is given by

$$P[n_{ij}] = \frac{e^{-E(n_{ij})/KT}}{\sum_{\substack{k:\text{ neighbor} \\ \text{of node } i}} e^{-E(n_{ik})/KT}} \tag{47.7}$$

where $E(n_{ij})$ is the energy gap $T[n_j] - T[n_i]$ if node n_j points to neighbor n_j and $E(n_{ik})$ is the energy gap for node n_j to point to all of its eight neighbors.

In our computer simulation, a random number is generated to see which direction a cell points to. We repeat this decision process for each sensor node in the lattice at each discrete time step. If we sweep the lattice for a sufficiently large number of times, then the fraction of times for sensor nodes pointing to a specific direction will be close to the calculated true probability.

To investigate how the kinetic factor tunes the overall macroscopic phase in the spin-glass model, we look at the relative probability of state A and state B:

$$\frac{P[A]}{P[B]} = e^{-D/KT} \tag{47.8}$$

If KT is much larger than the energy gap $(D = E(A) - E(B))$ between state A and state B, then the probability of taking either spin direction is approximately the same and the system is in a high-entropy state. If KT is much lower than D, then the sensor node is far more likely to be in a lower energy state. Generally speaking, low-temperature systems exhibit better routing performance in terms of hop distance. T is important, because the shortest path is not the only important criterion. A large T may reduce the power drain on choke points by taking longer routes; an extremely low T can protect the system by reducing oscillations in the system. Moreover, T can be specified on a per-region basis, allowing flexible control over the terrain. A sequence of ''snapshots'' of the system will be captured and displayed as a time-dependent process in our simulation animation. Our approach works differently from the standard statistical physics simulation technique, i.e. the Metropolis algorithm. However, the underlying idea is quite similar.

47.5.1.2 Spin-Glass Simulation Results

To quantify system adaptation we measure the mean distance from each node to the data sink. Figure 47.9 shows the mean number of hops versus generation number (time step) for a low-temperature system (Low T), high-temperature system (High T), and a system with a topological

© 2005 by Chapman & Hall/CRC

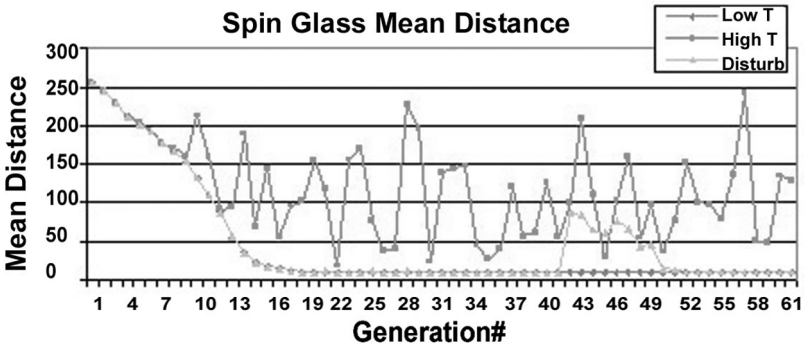


Figure 47.9. Spin-glass mean distance convergence.

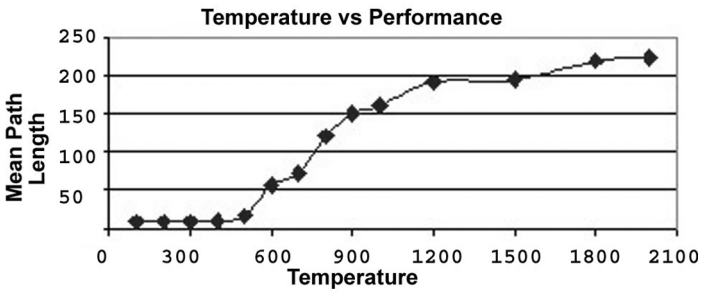


Figure 47.10. Effect of temperature on performance.

disturbance (Disturb). Topological disturbances correspond to the choke points in Figure 47.7 opening or closing. The system converges well when T is small, but not when T is large. The graph indicates that topological disturbances are accommodated after a number of fluctuating generations. This demonstrates that our system is capable of adapting to topological changes without the use of global information. Figure 47.10 further illustrates how mean distances are affected at various temperatures. We observe that there is an abrupt rise in mean hops if the temperature is raised above 500 K, indicating that 500 K acts as a critical temperature in phase change.

The amount of messages sent during the route establishment phase is quantified to evaluate the scalability of our routing model. We also study how the spin-glass model behaves under error conditions. Figure 47.11 demonstrates how system performance in terms of mean hops is affected by error conditions, which we consider to be when nodes randomly choose a spin direction instead of following the Boltzmann distribution function or when nodes send incorrect potential values to neighbors. It was observed that the system is very sensitive to these errors. The performance deteriorates drastically even if small percent of cells are in error condition. Figure 47.12 illustrates the communication cost versus error conditions. As expected, the amount of messages exchanged increases largely due to error messages diffusing throughout the system. We conclude that although the spin-glass model achieves high performance, the power consumption is high and the error tolerance is very limited.

47.5.2 Multi-Fractals

47.5.2.1 Adapting Multi-Fractals to WSN Routing

The classic irreversible fractal growth model for gas and fluid is called DLA, first introduced by Witten and Sander (WS) in the early 1980s. Beginning with one foreign seed, or even a line segment, a random

© 2005 by Chapman & Hall/CRC

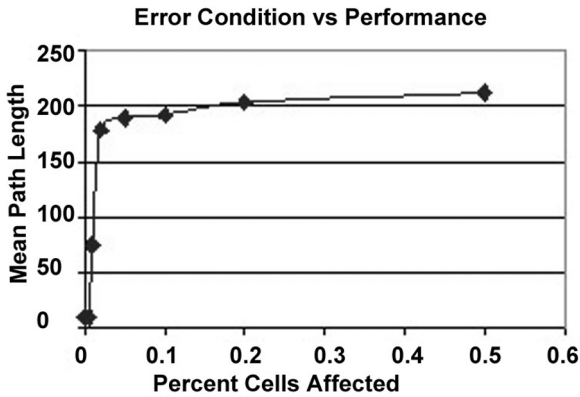


Figure 47.11. Error conditions versus performance.

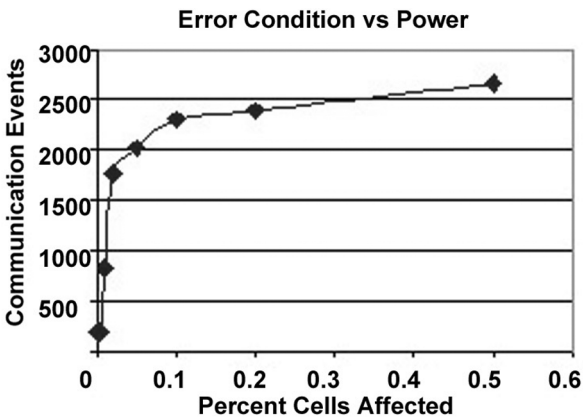


Figure 47.12. Error conditions versus power.

walk of gas or liquid particles becomes immobilized upon contact with the seed, if certain crystallization conditions are satisfied. Randomly diffusing particles keep sticking to each other, forming an aggregate. The structure of this fractal is affected by many factors, including crystallization growth inhibition exerted by the crystallization site prohibiting adherence by nearby particles. Interfacial surface tension and latent heat diffusion effects can physically explain this inhibition [7]. Such WS-like cluster examples can be found in metal electro-deposition experiments. Figure 47.13 shows an example of DLA.

DLA can be considered as a self-repelling random walk starting from the data sink. Let us look at the simple random walk in Figure 47.14: construct a connectivity matrix P whose i, j entry represents the probability of going from node i to node j:

$$P = \begin{bmatrix} 0 & 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1/2 & 0 \\ 1/3 & 0 & 1/3 & 1/3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{47.9}$$

If a mobile agent is initially at node B, then the initial state vector is $[0, 1, 0, 0]$. After the first step, the probability distribution of this agent is P_1:

$$P_1 = P_0 \times P = [1/2, 0, 1/2, 0] \tag{47.10}$$

© 2005 by Chapman & Hall/CRC

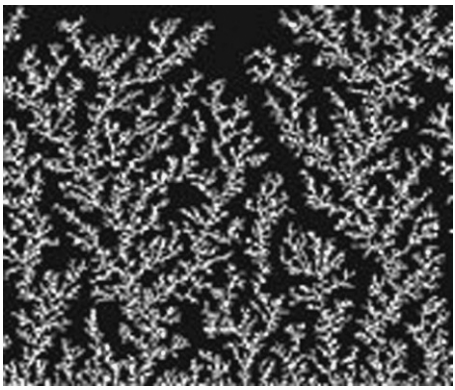


Figure 47.13. Example of DLA.

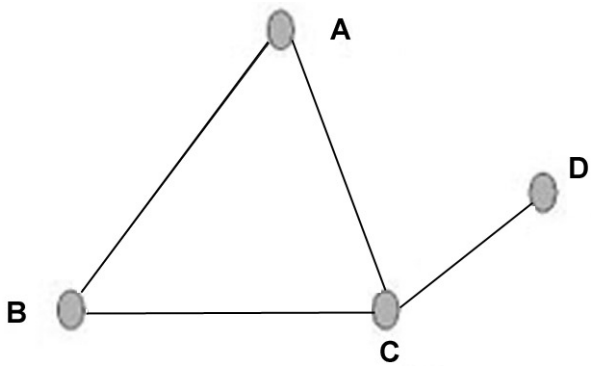


Figure 47.14. Random walk graph example.

We can easily get the state vector P_N after n discrete steps after P_0:

$$P_N = P_0 \times P^N \tag{47.11}$$

Equation (47.11) is actually a Markov chain. In the DLA model, a set of "stickiness" probabilities is specified based on the number of neighboring tree nodes. We can easily construct a connectivity matrix, as shown in matrix P, based on the specified probability set.

47.5.2.2 Multi-Fractal Simulation Results

In our multi-fractal routing model, the data sink is set to be the single foreign seed. Each sensor node vibrates in fixed coordinates of the lattice without randomly wandering around, as gas or fluid particles do; however, we do not exclude the mobility of sensor nodes, as our routing model can respond and accommodate topological change. A routing tree starts growing from the seed. A sensor node can possibly attach itself to the tree only if any tree nodes reach its neighborhood. Based on the number of neighboring immobilized tree nodes, a set of probabilities of joining the routing tree is specified. Theoretically speaking, nodes are less likely to join in the routing tree as the number of neighboring tree nodes increases. In other words, node stickiness decreases as the number of neighboring tree nodes increases, which is also known as the self-repelling effect. Depending on the different levels of self-repulsion we specified in the probabilities sets, the growth rate and the routing tree structure can be controlled.

© 2005 by Chapman & Hall/CRC

Generally speaking, a sparse tree with a high region coverage and grown in reasonable amount of time steps is desired. As previously stated, probabilities are the only way for us to control our routing tree. In order to select an ideal probabilities set for a good routing tree, a fitness function is constructed to evaluate the quality of the routing trees under different probability sets:

$$F_i = \frac{C_i}{T_i/b + N_i} \tag{47.12}$$

where C_i is the percentage region coverage, T_i are the discrete time steps, N_i is the number of tree nodes, and b is a constant used to normalize the time steps and the number of tree nodes. The higher the fitness value is, the better the routing tree is. Constant b actually represents our trade-off between sparsity and routing time. Actually, this fitness value is much like the fractional dimension we discuss in 47.2.2 to quantify the structure and quality of the fractal object.

Figure 47.15 shows the mean number of hops per generation number (time step) with and without topological disturbances (as for the spin-glass model in Figure 47.9). Communication cost and error tolerance are investigated, as was done on the spin-glass model. Further, malfunctioning nodes are not restrained by the desired multi-fractal behavior. Two principal malfunctions have been modeled: (1) faulty nodes have the same probability of joining the tree or not; (2) faulty nodes randomly choose a neighbor tree node to attach to.

Figure 47.16 shows how the error condition affects the routing performance. This graph indicates that there is a slight increase in the mean number of hops when random errors come into play; but,

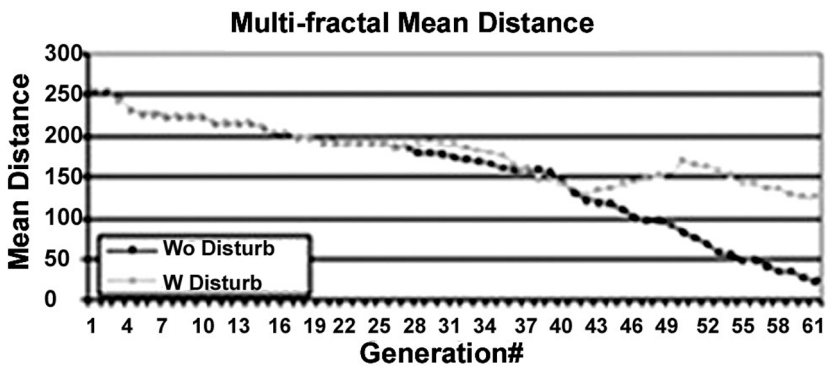


Figure 47.15. Multi-fractal mean distance convergence.

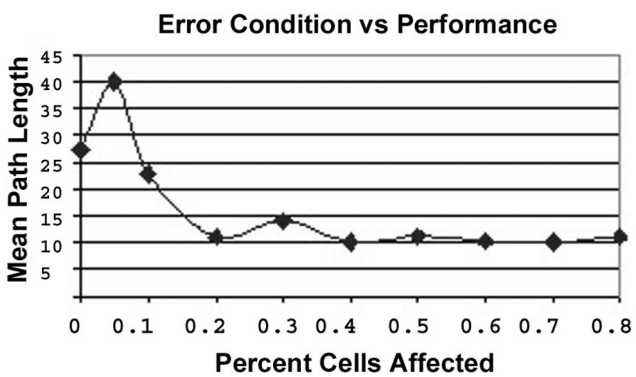


Figure 47.16. Multi-fractal effect of error condition on performance.

© 2005 by Chapman & Hall/CRC

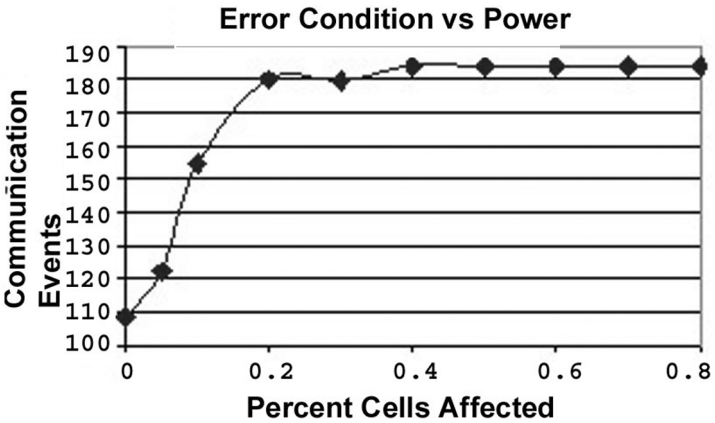


Figure 47.17. Multi-fractal effect of error condition on power.

once more errors begin to occur, the mean number of hops begins to drop down to almost half and stays steady, while the percentage of cells affected goes beyond 20%. We may then ask whether the error condition actually improves routing performance. The shorter mean hops come with the cost of denser trees. Recall that we want a sparse tree that covers most of the region. However, the final tree consists of approximately 75% more tree nodes than the original zero-error tree. The fitness of the error-conditioned tree is actually not ameliorated.

We also investigate how error conditions affect the cost of communication. Figure 47.17 demonstrates that error conditions incur considerably more communication events; nevertheless, the extra communication cost is much lower than that of the spin-glass model under the same error conditions. This indicates that the multi-fractal model is more error resilient in terms of performance and power.

47.6 Protocol Comparison and Discussion

Many routing protocols have been proposed for WSNs. The link state routing algorithm requires global knowledge about the network. Global routing protocols suffer serious scalability problems as the network size increases [8]. The DSDVR algorithm is an iterative, table-driven, and distributed routing scheme that stores the next hop and number of hops for each reachable node. The routing table storage requirement and periodic broadcasting are the two main drawbacks to this protocol [8]. In the dynamic source routing (DSR) protocol, a complete record of traversed cells is required to be carried by each data packet. Although no up-to-date routing information is maintained in the intermediate nodes' routing table, the complete cell record carried by each packet imposes storage and bandwidth problems. The AODVR algorithm alleviates the overhead problem in DSR by dynamically establishing route table entries at intermediate nodes, but symmetric links are required by AODV. Cluster-head gateway switch routing (CGSR) uses DSDVR as the underlying routing scheme to address the network hierarchically. Cluster-head and gateway cells are subject to higher communication and computation burden, and their failure can greatly deteriorate our system [9]. The greedy perimeter stateless routing algorithm claims to be highly efficient in table storage and communication overhead. However, it relies heavily on the self-describing geographic position, which may not be available under most conditions. In addition, the greedy forwarding mechanism may prohibit a valid path from being discovered if some detouring is necessary [10].

The spin-glass and multi-fractal models are related to the table-driven routing protocols by establishing routes from every cell to data sink(s). These protocols ensure timely data transmission on demand without searching for the route each time. The ant pheromone model, as discussed in Chapter 46, is related to packet-driven protocols. Ants can be viewed as packets traversing from data sources to

© 2005 by Chapman & Hall/CRC

data sinks. All of the models we presented are decentralized, using only local knowledge at each node. They adapt dynamically to topological disturbances (path loss). Storage requirements for the routing tables of spin-glass and multi-fractal models are low compared with most other protocols, while the ant pheromone's storage requirements are even lower than these two.

The temporally ordered routing algorithm is a source-initiated and distributed routing scheme that shares some properties with the spin-glass model. It establishes an acyclic graph using a height metric relative to the data sink and also has local reaction to topological disturbances [9].

The kinetic factor in our spin-glass model and the frequency of ant generation in the ant pheromone model provide the system with flexibility in controlling routing behaviors under various conditions. Route maintenance overhead is moderately high for the spin-glass model.

The multi-fractal approach, as a probabilistic space-filling curve, has very light computation and communication load, and overhead is saved in route discovery and maintenance. This is at the cost of a higher distance to the data sink(s). Route maintenance overhead for the pheromones is low due to the reduced number of nodes involved in each path. Since the multi-fractal model strives to cover the sensor field by using as few cells as possible, the sparse routing tree conserves energy. The shortest routes to the data sink are not found using the multi-fractal model.

On the other hand, the spin-glass model is more sensitive to internal errors, since any possible error may diffuse throughout the network. The multi-fractal and ant pheromone models are very resistant to internal errors. The time required for the ant pheromone algorithm to converge to a steady state is much longer than required by the other two adaptations. For applications requiring short data paths, the spin-glass model is preferred. For overhead-sensitive applications that require quick deployment, the multi-fractal model is a better candidate. If error resilience and low overhead are the principle requirements, then the ant pheromone model is appropriate. Hybrid methods, or switching between methods at different phases, may be useful.

Acknowledgment and Disclaimer

This research is sponsored by the Defense Advance Research Projects Agency (DARPA), and administered by the Army Research Office under Emergent Surveillance Plexus MURI Award No. DAAD19-01-1-0504. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the sponsoring agencies.

Reference

[1] Fitzpatrick, R., Lecture notes of Introduction to Computational Physics, 2002.
[2] Wolfram, S., *A New Kind of Science*, Wolfram Media, 2002.
[3] Major, A., and Lantsman, Y., *Part I Actuarial Application of Multifractal Modelling*, 2001 Data Management, Quality, and Technology Program Sponsored by the Casualty Actuarial Soceity, 2001.
[4] Preston, K. Jr. and Duff, M.J.B., *Modern Cellular Automata Theory and Applications*, Plenum Press, 1984.
[5] Akyildiz, I.F. *et al.*, Wireless sensor networks: a survey, *International Journal of Computer and Telecommunications Networking*, 38(4), 393, 2001.
[6] Gottschalk, T. and Davis, D., Hrothgar Project, Center for Advanced Computing Research at Caltech, 2000.
[7] Gaylord, R.J. and Nishidate, K., *Modeling Nature Cellular Automata Simulations with Mathematica*, Springer-Verlag, New York, 1996.
[8] Kurose, J.F. and Ross, K.W., *Computer Networking a Top-Down Approach Featuring the Internet*, AW Higher Education Group, 2003.
[9] Royer, E.M., A review of current routing protocols for *ad hoc* mobile wireless networks, *IEEE Personal Communications*, 6(2), 46, 1999.

© 2005 by Chapman & Hall/CRC

[10] Karp, B. and Ung, H.T.K., Greedy perimeter stateless routing for wireless networks, in *Proceedings of the 6th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, 2000.

[11] Brochmann, H., Introduction to fractal geometry, http://www.saltspring.com/brochmann/math/Fractals/fractal-0.00.html (last accessed on 7/26/2004).

© 2005 by Chapman & Hall/CRC

48

Collective Intelligence for Power-Aware Routing in Mobile *Ad Hoc* Sensor Networks

Vijay S. Iyer, S.S. Iyengar, and N. Balakrishnan

48.1 Introduction

Life can be defined as a state of functional activity and continual change peculiar to animals and plants before death. It can also be classified as something capable of reproducing itself, capable of adapting to an environment, and also capable of independent actions not decided by some external agent.

48.1.1 Artificial Life

Artificial life, or Alife, is the study of nonorganic organisms, of lifelike behavior beyond the creation of nature. In artificial life, the environment is originally created by humans inside a computer. The rules of life are fairly universal and apply even out of the natural setting. A set of rules is created for the creatures to follow. The creatures interact with each other and arrive at a solution (a state of global optimization).

One of the important characteristics of both life and Alife is emergence. Emergence is *something more than the sum of the parts*. In Alife simulations, global patterns emerge as a result of the behavior of the individual entities and interaction amongst them. Alife adopts a bottom-up approach, in which only the behavior of the lower level units are programmed, and the higher level *collective behavior* is obtained from the interactions between these units [1]. In nature, many organisms exhibit collective behavior, including flocking by birds, schools of fish, nest building by social insects, and foraging by ants.

48.1.2 Stigmergy

An important collective behavior mechanism that makes insects perform well in a large variety of tasks is called stigmergy. Stigmergy is a form of indirect communication through the environment.

© 2005 by Chapman & Hall/CRC

Stigmergy is of two forms:

1. In the first form, the physical characteristics of the environment may change as a result of carrying out some task-related action, such as digging a hole by an ant, or adding a ball of mud to a growing structure by a termite. Subsequent perception of the changed environment may cause the next ant to enlarge the hole, or the termite to deposit its ball of mud on top of the previous structure. This type of influence has been called sematectonic [2].
2. In the second form, the environment is changed by depositing something that makes no direct contribution to the task, but is used solely to influence subsequent behavior, which is task related. This is sign-based stigmergy, which is highly developed in ants and other social insects. Ants use a special type of hormone, known as a pheromone, to provide a sophisticated signaling system.

48.1.3 Trail Laying by Ants

As mentioned earlier, ants have a peculiar property of laying pheromone trails while going in search of food or returning back to their nests. Ants get attracted to the pheromone deposits depending on the trail strength. A pheromone is a hormone, which evaporates and diffuses into the atmosphere. At any given time the strength of the trail encountered by another ant is a function of the original trail strength and the time since the trail was laid.

Figure 48.1 shows more than one possible route between the nest and the food source. Initially, an ant arriving at the junction makes a random decision with a probability of 0.5 of choosing one route over the other. Now, assume that there are two ants in search of food and two more returning from the food source towards the nest. Let us assume that the ants in both the pairs chose to go by different routes.

After some time, the ants that chose the shorter path reach their destination first, as depicted in Figure 48.2. We see that the pheromone concentration in the shorter path is more than in the

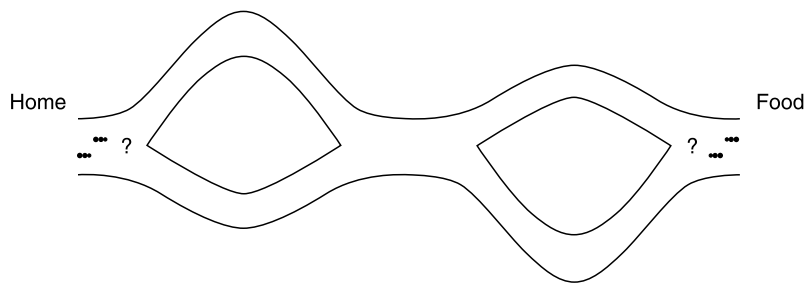


Figure 48.1. Ants have to make a decision.

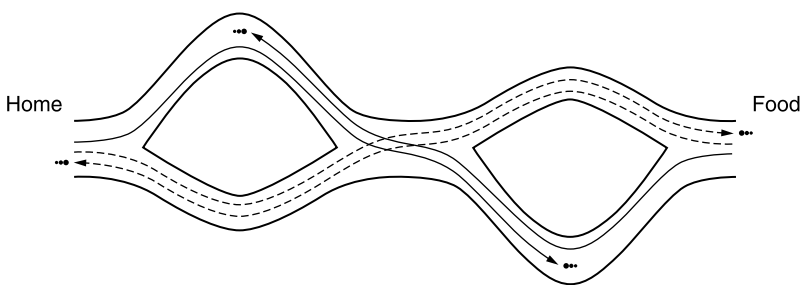


Figure 48.2. Ants laying pheromone trails.

© 2005 by Chapman & Hall/CRC

longer path. The next ant arriving at the junction would choose the shorter route with a higher probability for reaching its destination. This, in turn, enhances the pheromone concentration in the shorter route, causing more and more ants to traverse the shorter route. As fewer ants travel through the longer path, and the existing pheromone slowly evaporates, the trail in the longer path weakens and eventually disappears.

There is a kind of reinforcement learning that takes place in finding the optimal path. Overtraining raises two problems:

1. *Blocking problem.* This occurs when a route previously found by ants is no longer available. It can take a long time for ants to find a new route, because the trails leading to the blocking route are sometimes very strong.
2. *Shortcut problem.* This occurs when a new or shorter route suddenly becomes available. In this case, the shorter route may not be easily found, because the existing trails are so strong that almost all the ants choose them.

Please refer [7] to [27] for details on computational aspects of collective intelligence.

48.2 Artificial Ants for Routing

Below, we describe a technique derived from the collective behavior of ants for power-aware routing in mobile *ad hoc* networks. We have used specialized packets (artificial ants) to find the shortest (at least shorter) routes in a wireless network. These have already been made in telecommunication networks successfully for routing [3] and congestion control [4].

48.2.1 Routing Table

The routing table used in this method is inspired from *Ant Based Control* [3] and the *AntNet* [4] algorithms. The routing table assumes the structure of a probability table. The table gives the probability of choosing a neighbor for a given destination. For a network with n nodes the probability table in a particular node with m neighbors would contain $(n-1) \times m$ entries.

A typical routing for a node would look like the one shown in Table 48.1, which indicates the probabilities of choosing a route. The probabilities of the table sum up to 1.0. Updating the probabilities of the routing table represents the act of laying pheromones. When an ant reaches a node, it consults with the routing table, updates it, and chooses the next hop based on the random decision, but depending on the probabilities in the table. For example, consider an ant reaching node 1, which has the routing table exactly as given in Table 48.1. The ant's is destination is node 3, say. The ant will choose the next hop as node 2 with a probability of 0.45 or node 4 with a probability of 0.55 after consulting the row corresponding to nodes 3 in the routing table.

48.2.2 Routing of Data Packets

Data packets move according to the entries in the routing table corresponding to their destination. Data packets move independent of ants. To determine the next hop for a data packet with a given

Table 48.1. Probability of choosing a route

Destination	Next Hop	
	Node 2	Node 4
2	0.95	0.05
3	0.45	0.55
4	0.02	0.98

© 2005 by Chapman & Hall/CRC

destination, the row corresponding to the destination is looked up in the routing table. The neighbor with highest probability becomes the next hop.

48.2.3 Saturation Value

To overcome the blocking and shortcut problem (see Section ??), freezing of routes should be avoided. Freezing of routes takes place when the probability of choosing the next hop becomes unity for a particular neighbor and zero for others. This situation can be overcome by incorporating a saturation value above which none of the probabilities is allowed to rise. This allows a finite number of ants to still traverse other routes even if the best route has been found out.

48.2.4 Modification of Probabilities

As described in Section 48.2.4.1, δp depends on either the age of ants or the hop count and the energy left in the previous node. The prime requirement is to find methods for encouraging ants to establish shorter routes in the network. The measures for finding shorter routes are the age of the ant or the number of hops the ant has covered. Since maintaining time in mobile *ad hoc* networks is difficult, the hop count was used as a measure.

48.2.4.1 Hop Count

Since routing in mobile adhoc networks (MANETs) is a multihop scenario, hop count can be used as a measure of distance between the source and the destination. Using hop count, δp_{h} can be computed using

$$\delta p_{\mathrm{h}} = \frac{h_{\mathrm{f}}}{H} + h_{\mathrm{c}} \tag{48.1}$$

where H is the number of hops (nodes), including the source, that an ant has traveled, h_{f} is the hop factor (typically 0.2–0.3), and h_{c} is a constant (typically 0.01).

48.2.4.2 Power-Aware Routing

Energy in a mobile device is a scarce resource, so the need for its conservation arises. Here, a technique is presented to improve the energy utilization of the network. It is a well-known fact that if energy is tapped from a source like a battery in chunks, then the battery life extends [5].

The life of a network depends on how many connections are up at any given time. Figure 48.3 shows a four-node wireless network. Let us assume the case of normal routing without power awareness. The ants establish the route from node 1 to node 3 via node 4. After some time due to packet movement

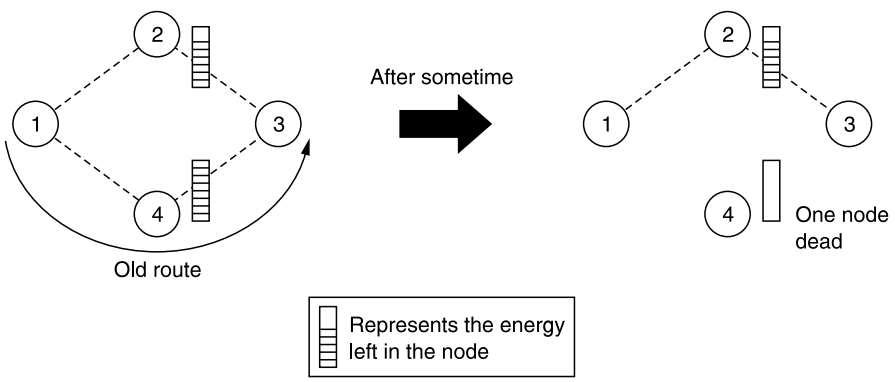


Figure 48.3. Normal routing without power awareness.

© 2005 by Chapman & Hall/CRC

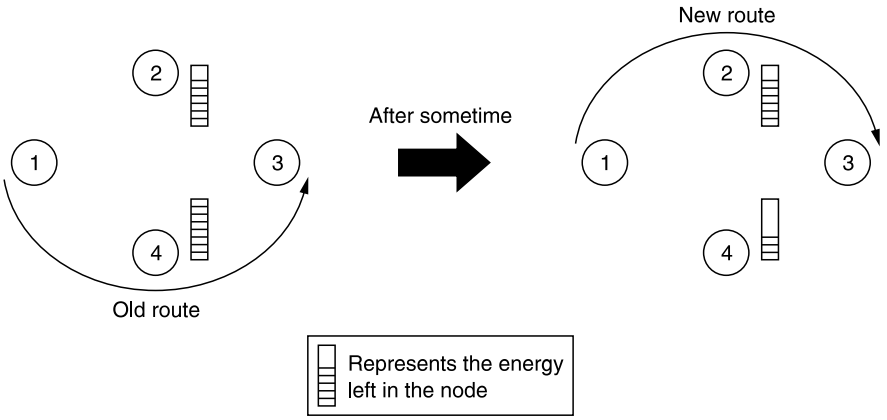


Figure 48.4. Change in route after some time due to power awareness in routing.

through node 4, energy in node 4 decreases, eventually leading to its death. After the death of node 4, node 2 is selected as a hop for routing data packets from node 1 to node 4. At this point in time, only two routes are up.

Figure 48.4 shows a way to improve the longevity of the network. Let us assume that the ants have established a route between node 1 and node 3 via node 4. Owing to movement of packets, the energy in node 4 decreases. Ants, after sensing a decrease in energy in node 4, change the route, and establish a new route from node 1 to node 3 via node 2. In this way the energy from the nodes is not tapped continuously; instead, the nodes are given time to charge themselves. And as we know, if the energy is not drawn continuously from a source, then its life increases [5].

Improving the longevity of a network by incorporating a power-awareness term in δp causes the links in the network to be up for a longer duration of time. δp_{p} is given by

$$\delta p_{\mathrm{p}} = \frac{1}{a(E/E_{\max})^2} \tag{48.2}$$

where E is the energy left in the previous hop at the time the ant was being transmitted, $E_{\max}$ is the maximum energy a device has at the time of initialization, and a is a constant (typically around 2–2.5).

48.2.5 Algorithm

1. Ants are launched from every node with randomly selected destinations.
2. The interval after which the ants are released is not fixed, but is randomly selected between a fixed range.
3. Ants move randomly in the network but, depending on the probabilities in the routing table, move towards their destination (see Section 48.2.1).
4. Ants modify the probabilities in the routing table for the location they were launched from, by increasing the probability of selection of their previous node by subsequent ants. The increase in probability is given by

$$p_{\mathrm{prev}} = \frac{p_{\mathrm{oldp}} + \delta p}{1 + \delta p} \tag{48.3}$$

where p_{prev} is the new probability corresponding to the previous node, p_{oldp} is the original probability and δp is the increase in probability. The other entries in the table are decreased

© 2005 by Chapman & Hall/CRC

correspondingly, as given by

$$p_{\text{other}} = \frac{p_{\text{oldo}}}{1 + \delta p} \tag{48.4}$$

where p_{other} is the new probability corresponding to other neighboring nodes and p_{oldo} is the original probability of the other nodes.

5. The increase in probabilities is a decreasing function of the hop count/age of ant, and energy left in the previous node and of the original probability (see Section 48.2.4).
6. To avoid freezing of trails, some saturation value (<1.0) is maintained, above which none of probabilities in the routing table is allowed to rise (see Section 48.2.3).

48.3 Results

The suggested algorithm was simulated for a mobile *ad hoc* network working on the 802.11 protocol. The topology of the network was generated randomly and, to add mobility to the network, nodes were moved at each time step of the simulation.

The following assumptions were made while designing the model:

1. There is no multi-path fading or distortion of the signal.
2. The coverage is perfectly circular.
3. The transmission takes place only at 1 Mbps.
4. The nodes were assumed to be situated in a two-dimensional plane.
5. There is no bit-error occurring in the transmission due to the medium, in case there is no other simultaneous transmission.

The energy consumption model for the Lucent WaveLAN PC card was used [6]. The typical energy dissipation during sending and receiving data is given in Table 48.2.

48.3.1 Route Establishment

The simulation results have been produced here. Figures 48.5 and 48.6 show the number of ants and the time required to establish routes in a network with respect to the size of the network. The route establishment was considered complete when there exists a unique path (when at least a path exists) between each pair of nodes in the network, which is at the maximum 10% longer than the shortest available path.

Table 48.2. Energy dissipation of the Lucent WaveLAN PC card

	μW-s/byte	μW-s
Point-to-point send	$1.9 \times size$	+454
Broadcast send	$1.9 \times size$	+266
Point-to-point recv.	$0.50 \times size$	+356
Broadcast recv.	$0.50 \times size$	+56
	Nondestination $n \in S, D$	
Promiscuous recv.	$0.39 \times size$	+140
Discard	$-0.61 \times size$	+56
	Nondestination $n \in S, n \notin D$	
Promiscuous recv.	$0.54 \times size$	+66
Discard	$-0.56 \times size$	+24
Idle (*ad hoc*)	843 mW	
Idle (BSS)	66 mW	

© 2005 by Chapman & Hall/CRC

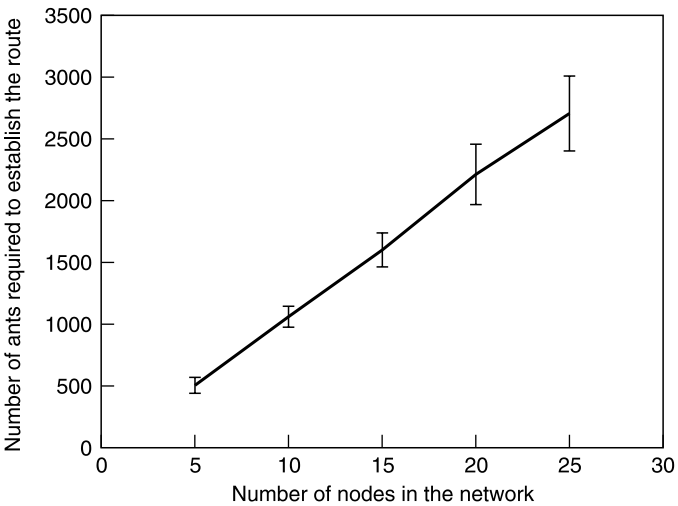


Figure 48.5. Number of ants versus size of network.

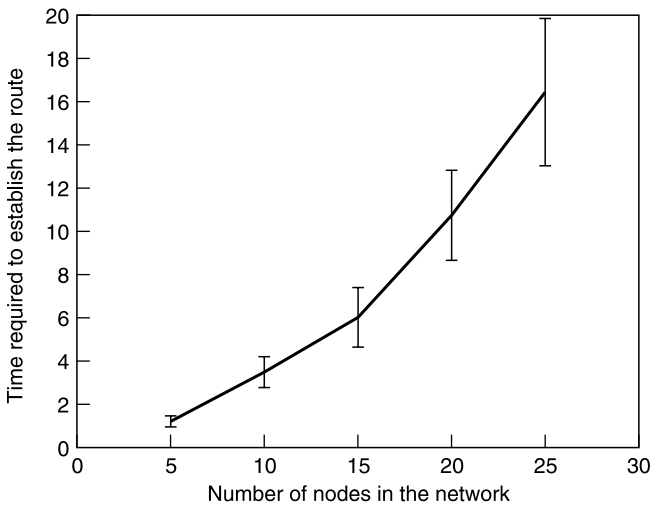


Figure 48.6. Time required versus size of network.

48.3.2 Energy Distribution

The longevity of a network depends on the number of critical nodes present in it. Critical nodes may be defined as those nodes that have much less energy and are susceptible to die. Figures 48.7 and 48.8 show the graphs between number of nodes and percentage decrease in energy. As is clear from the figures, the use of power awareness in routing has caused this number to decrease, thus enhancing the longevity of the network.

Figure 48.9 illustrates the increase in energy awareness due to the new routing methodology. Figure 48.9(a) shows the average energy of all the nodes in a 30-node network with and without power-awareness. Figure 48.9(b) shows the corresponding standard deviation.

© 2005 by Chapman & Hall/CRC

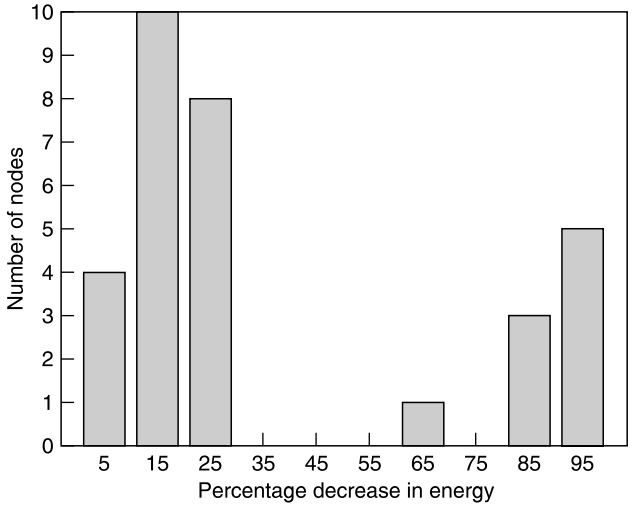


Figure 48.7. Illustration of the percentage decrease in energy with the number of nodes (describes longevity of the network).

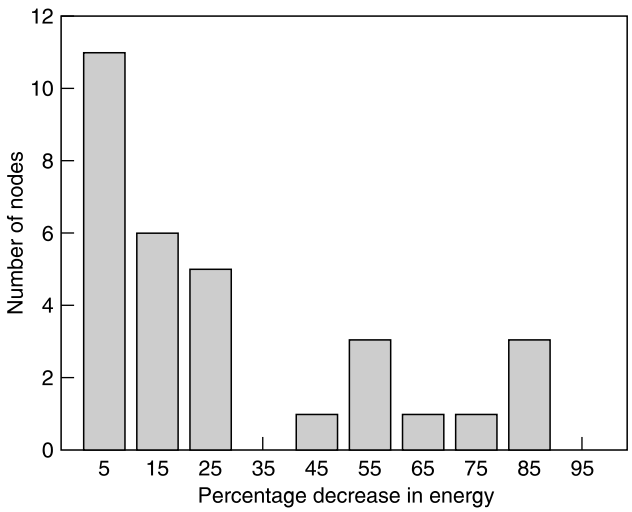


Figure 48.8. Illustration of critical nodes that have much less energy and susceptible to death.

48.3.3 Energy Access Pattern

The access patterns for a particular node following simulations performed over a network of ten nodes are shown in Figures 48.10(a) and (b). From the energy access patterns, it is clear that the energy consumption from that node has lowered over a period of time with power awareness accounted for, which would improve the longevity of the network.

48.3.4 Effect of Noise

As mentioned in Section 48.2.4.2, we set a saturation value for the probabilities to avoid freezing of ant trails (routes). Figure 48.11 shows the average and standard deviation of energies of a network with and

© 2005 by Chapman & Hall/CRC

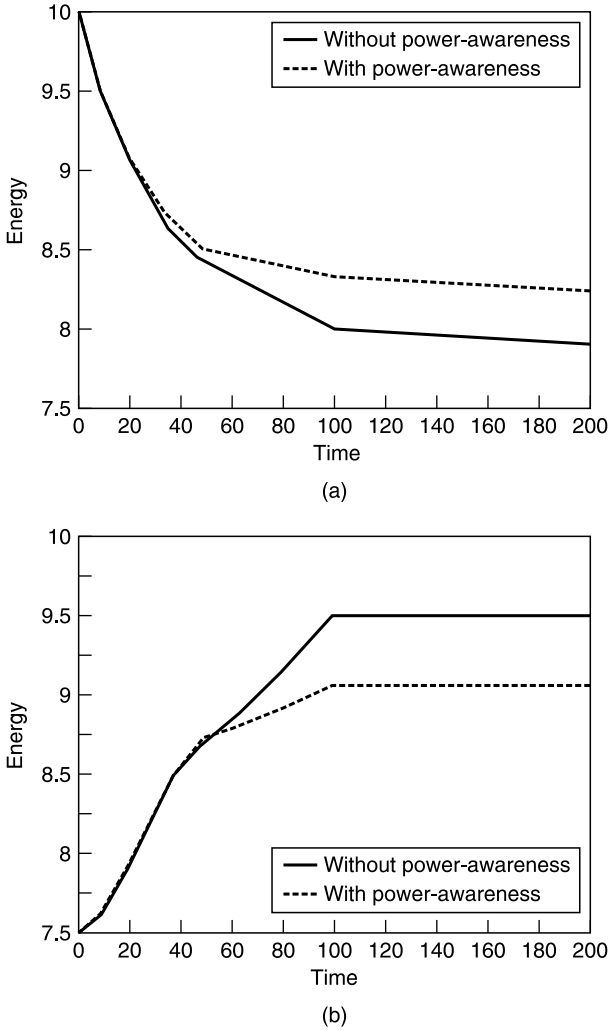


Figure 48.9. Average and standard deviation of energies of a network of 30 nodes (a) with and (b) without power awareness factor.

without saturation value. Clearly, the saturation value has given a significant decrease in power consumption of the network.

48.4 Conclusion

Emergent behavior gives rise to collective intelligence or the swarm intelligence, which has been used here to identify routes in mobile *ad hoc* sensor networks. Mobile *ad hoc* sensor networks have tremendous application in surveillance, military and civil communications, personal communication, and many other networks of prime importance. Routing, being a problem of global optimization, is highly complex. In mobile *ad hoc* sensor networks the problem manifolds in complexity due to the mobility of the nodes. In this chapter we have used specialized packets known as ants to establish routes. We also address the issue of scarcity of energy in mobile devices by making the algorithm power-aware. The algorithm is highly adaptive and scalable.

© 2005 by Chapman & Hall/CRC

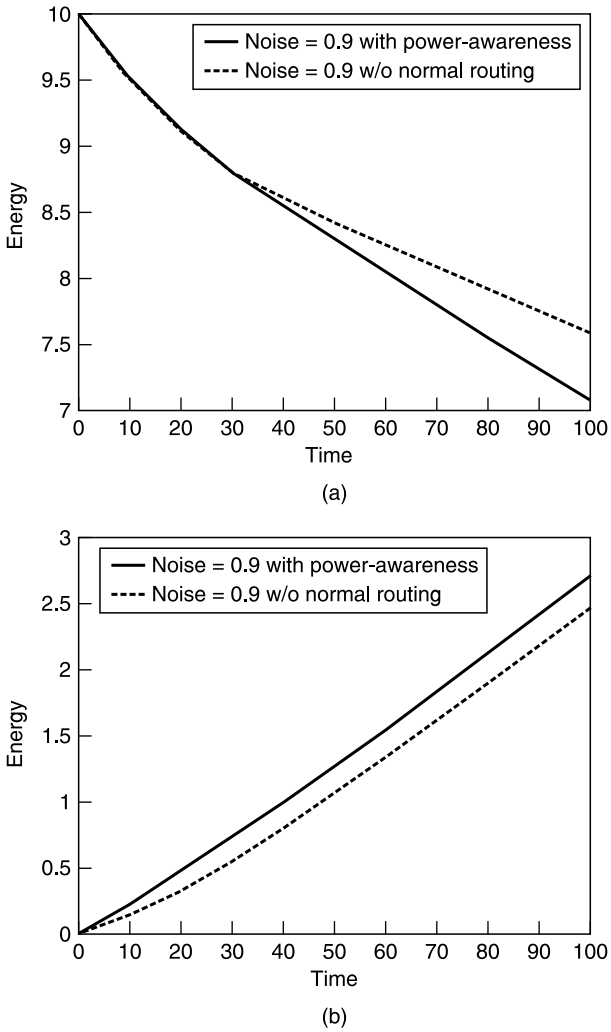


Figure 48.10. Energy access pattern: (a) without power awareness; (b) with power awareness.

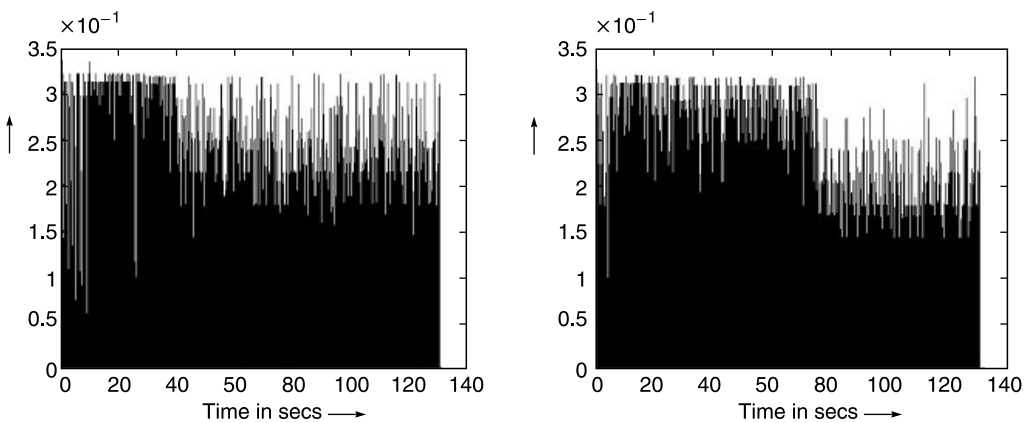


Figure 48.11. Average (a) and standard deviation (b) of the energies of a network consisting of 30 nodes with and without power-awareness factor when the saturation value is set to 0.9.

© 2005 by Chapman & Hall/CRC

References

[1] Kawata, M. and Toquenaga, Y., From artificial individuals to global patterns, *TREE*, 9(11), 417, 1994.

[2] Bonabeau, E., Theraulaz, G., Arpin, E., and Sardet, E., The building behaviour of lattice swarms, *Artificial Life IV*, Brooks, R., and Maes, P. (eds), MIT press, 1994, pp. 307–312.

[3] Di Caro, G. and Dorigo, M., AntNet: a mobile agents approach to adaptive routing, *Technical Report IRIDIA/97-12, Université Libre de Bruxelles*, Belgium, 1997.

[4] Schoonderwoerd, R., Collective intelligence for network control, M.S. Thesis, Delft University of Technology, Faculty of Technical Informatics, May 1996.

[5] Balakrishna, J. *et al.*, Performance analysis of battery power management schemes in wireless mobile devices, in *IEEE WCNC'2002,* Orlando, March 2002.

[6] Feeney, L.M., An energy consumption model for performance analysis of mobile *ad hoc* networks, *Journal of Mobile Networks and Applications*, 6(3), 2001.

[7] Alcherio, M., Collective complexity out of individual simplicity, *Invited book review on Swarm Intelligence: From Natural to Artificial Systems*, by Bonabeau, E. *et al.*, *Artificial Life*, 7(3), 315, 2001.

[8] Arabshahi, P. *et al.*, Adaptive routing in wireless communication networks using swarm intelligence, in *Proceeding of 19th AIAA International Communications Satellite Systems Conference,* Toulouse, France, 17–20 April 2001.

[9] Beckers, R. *et al.*, Trails and U-turns in the selection of a path by the ant *Lasius niger*, *Journal of Theoretical Biology*, 159, 397, 1992.

[10] Beckers, R. *et al.*, From local actions to global tasks: stigmergy and collective robotics, *Artificial Life IV, Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, Brooks, R.A. and Maes, P. (ed.), MIT Press, 1994, 181.

[11] Barán, B. and Sosa, R., AntNet: routing algorithm for data networks based on mobile agents, *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial*, (12), 75, 2001.

[12] Bonabeau, E. *et al.*, Ant colony optimization: a new meta-heuristic, in *Proceedings of 1999 Congress on Evolutionary Computation*, July 1999, 1470.

[13] Bonabeau, E. and Theraulaz, G., Swarm smarts, *Scientific American*, March, 73, 2000.

[14] Colorni, A. *et al.*, The Ant System: optimization by a colony of cooperating agents, *IEEE Transactions on Systems, Man, and Cybernetics*, *Part B*, 26(1), 1, 1996.

[15] Di Caro, G. and Dorigo, M., AntNet: a mobile agents approach to adaptive routing in communication network, in *9th Dutch Conference on Artificial Intelligence (NAIC '97)*, Antwerpen (BE), November 12–13, 1997.

[16] Di Caro, G. and Dorigo, M., AntNet: distributed stigmergetic control for communications networks, *Journal of Artificial Intelligence Research*, 9, 317, 1998.

[17] Di Caro, G. and Dorigo, M., Ant colony routing, in *PECTEL 2 Workshop on Parallel Evolutionary Computation in Telecommunications,* Reading, England, April 6–7, 1998.

[18] Di Caro, G. and Dorigo, M., Adaptive learning of routing tables in communication networks, in *Proceedings of Italian Workshop on Machine Learning*, Torino (IT), December 9–10, 1997.

[19] Di Caro, G. and Dorigo, M., Mobile agents for adaptive routing, in *Proceedings of 31st Hawaii International Conference on System Sciences,* IEEE Computer Society Press, Los Alamitos, CA, 1998, 74.

[20] Feeney, L.M. and Nilsson, M., Investigating the energy consumption of a wireless network interface in an *ad hoc* networking environment, in *IEEE Infocom 2001*, Anchorage, AK, April 2001.

[21] IEEE 802 LAN/MAN Standards Committee, Wireless LAN medium access control (MAC) and physical layer (PHY) specifications, IEEE Standard 802.11, June 1999.

[22] Kassabalidis, I. *et al.*, Swarm intelligence for routing in satellite and sensor networks, in *NASA Earth Science Technology Conference*, College Park, MD, August 28–30, 2001.

[23] Kassabalidis, I. *et al.*, Swarm intelligence for routing in communication networks, in *IEEE GlobeComm*, November 2001.

© 2005 by Chapman & Hall/CRC

[24] Langton, C.G., Artificial life, in *Proceedings of an Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems*, Los Alamos, New Mexico, Langton, C.G. *et al.* (eds), Addison Wesley, 1987.
[25] Langton, C.G., Preface, Artificial Life II, in *Proceedings of the Workshop on Artificial Life*, Santa Fe, New Mexico, Langton, C.G. *et al.* (eds), Addison Wesley, 1990.
[26] Schoonderwoerd, R., *et al.*, Ant-based load balancing in telecommunications networks, HP Labs Technical Report, HPL-96-76, May 21, 1996.
[27] Schoonderwoerd, R. *et al.*, Ant-like agents for load balancing in telecommunications networks, in *Proceedings of 1st ACM International Conference on Autonomous Agents*, Marina del Rey, CA, USA, February 5–8, 1997, 209.

© 2005 by Chapman & Hall/CRC

49

Random Networks and Percolation Theory

R.R. Brooks

49.1 Notation

a	edge availability $(1-f)$
$\hat{a}$	effective edge availability including graph redundancy effects
$\hat{a}_r$	effective edge availability including effects due to redundant paths of r or fewer hops
b	constant factor in a scale-free probability distribution
C	cluster coefficient
c	number of nodes in fully connected components of a connected caveman small-world graph
d	degree of a node
d_{ave}	average node degree
E	set of edges defining a graph
e	number of rewired edges in a small-world graph
f	probability of link failure $(1-a)$
h	the expected number of hops between two nodes in a graph
γ	scaling factor in a small-world graph
L	graph connectivity matrix
$l_{i,j}$	element i, j of L
$M[h]$	mutuality factor for r hops
n	number of nodes in the graph
p	probability of an edge between two nodes for Erdös–Rényi graph
$P_{x,\ldots,\neg y,\ldots,z}$	shorthand for $P[\{q_x \wedge \ldots \wedge \neg\, q_y \wedge \ldots \wedge q_z\}]$
q_r	expected number of nodes reachable in r hops
$\{q_r\}$	set of nodes reachable in r hops
V	set of vertices defining a graph
z_r	expected number of nodes first reachable in r hops
$\{z_r\}$	set of nodes first reachable in r hops

© 2005 by Chapman & Hall/CRC

49.2 Background

Random graph theory originated with the seminal work by Erdös and Rényi in the 1950s. Until then, graph theory analyzed either specific graph instances or deterministically defined graph classes. Erdös and Rényi considered graph classes where the existence of edges between nodes was determined probabilistically. (For graph theory background, see Section 49.3.) Their results were theoretically interesting and found applications in many practical domains [1].

Erdös and Rényi used the same probability value to assign edges between any two nodes in the graph. As an extension to this, in the 1990s Strogatz and Watts studied small-world graphs [2]. The term "small world" originates with Milgram's six degrees of separation model of social networks created in the 1960s. Strogatz and Watts's work considers networks where the probability of edges existing between nodes is not uniform. They were specifically interested in clustered graphs, where edges are more likely to exist between nodes with common neighbors. To study this phenomenon, they defined classes of pseudo-random graphs. These graphs combine a deterministic structure and a limited number of random edges. Their results have been used to analyze both social networks and technical infrastructures.

An alternative approach to studying similar systems has been proposed by Barabási [1]. His group considered the probability distributions of graph node degree found in graph models of existing systems. This analysis shows that the probability of a node having degree d follows an inverse power law (i.e. is proportional to $d^{-\gamma}$ where γ is a constant). They also explain how this property can emerge from positive feedback in evolving systems. These models appear to be appropriate for studying a wide range of natural and large-scale technical systems. Important results from this model include quantification of the dependability of the Internet [3] and analysis of computer virus propagation [4].

Random graph concepts are also widely used in percolation theory [5]. Percolation theory studies flow through random media. The model of random media is usually built from a regular tessellation of an n-dimensional space. Edges may or may not exist between neighboring vertices of the tessellation with a uniform probability. Applications of percolation theory include oil extraction. We consider this model as an example of sensor networks with a planned wireless infrastructure.

Another random network model, given by Krishnamachari *et al.* [6], is used to study *ad hoc* wireless networks. A set of nodes is randomly distributed in a two-dimensional region. Each node has a radio with a given range r. A uniform probability exists (in the Krishnamachari *et al.* [6] model the probability is unity) for edges being formed between nodes as long as they are within range of each other. This network model has obvious practical applications. Many of its properties resemble those of Erdös–Rényi graphs, yet it also has significant clustering like the small-world model. We use this model when analyzing *ad hoc* wireless systems.

Many network systems that would otherwise be intractable can be analyzed using random graph abstractions. This chapter is inspired by the design of adaptable peer-to-peer (P2P) computer networks, which are particularly suited to implementing sensor networks. Consider a P2P network with no centralized design, control, or plan. Nodes enter and leave the system of their own volition [7]. In P2P networks, all participants function simultaneously as both client and server, which is why they are sometimes called "servents." The existence or nonexistence of an interaction between two nodes cannot be known in advance, making the random graph model an appropriate one for some aspects of the system.

P2P networking gained recognition with the Napster and Gnutella implementations. Napster is a scalable approach to file dissemination. On connecting to Napster, each user workstation uploaded a list of the local filenames to the Napster server. Filenames were on the order of tens of bytes. The files themselves were usually multi-megabyte MP3 music files. To retrieve a file, the user queried Napster's index and received a list of potential offerors. File exchanges occurred between individual nodes distributed at random across the Internet. This was very efficient, but with a single point of failure. A court order stopped the central index, effectively destroying the entire system.

© 2005 by Chapman & Hall/CRC

Gnutella's distributed design is extremely robust. A Gnutella network of n nodes has n indexes. Each node keeps track of locally stored files. The network has no single point of failure. To stop the Gnutella service, it would be necessary to stop every node running Gnutella on the Internet. This is a desirable survivability property. On the other hand, a global search of a Gnutella network involves flooding the network with search packets. This is inefficient and scales poorly [8]. A relatively small number of concurrent requests can use all available network bandwidth, creating an unintentional internal denial-of-service (DoS) attack.

We studied this trade-off between efficiency and robustness by generalizing the P2P design problem of Kapur *et al.* [9]. The paper shows how to determine the proper number of indexes (in the range 1 to n) and packet time-out values to support desired levels of robustness and quality of service (QoS). An essential part of the analysis was estimating the expected number of hops between nodes. To do so, we relied heavily on the random graph analysis techniques described in this chapter.

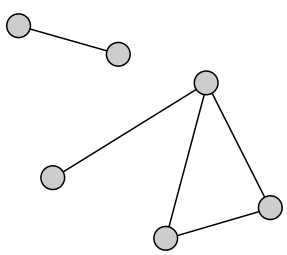
We describe different classes of random graphs that P2P sensor networks can form using different connection strategies. Analysis of the random graph models provides insights for which strategy to use. The probabilistic connectivity matrices described here provide a uniform representation of the random graph classes considered and explain many of their more subtle properties. A method is given for identifying phase changes in system behavior and associated critical points. An algorithm is given for creating scale-free graphs with an arbitrary scaling factor. Methods are given for computing conditional probabilities in random graph systems.

49.3 Graph Theory

Traditionally, the tuple $[V, E]$ defines a graph. V and E are sets of vertices and edges respectively. Each edge e is defined by (i, j), where i and j are the two vertices connected by e. Unless specified otherwise, we will consider only undirected graphs where $(i, j) = (j, i)$. Directed graphs (di-graphs) exist where $(i, j) \neq (j, i)$. An edge (i, j) is incident on vertices i and j. We do not consider multi-graphs where multiple edges connect the same end points. The terms vertex and node will be used interchangeably. Similarly, edge and link are used synonymously.

The degree of a node is the number of edges incident on the node. For directed graphs, there are also the concepts of in-degree (out-degree) for the number of edges leaving (joining) the node. A graph is called regular when all nodes in the graph have the same degree.

Many data structures can be used as practical graph representations. Common representations can be found in Aho *et al.* [10]. For example, a graph where each node has at least one incident edge can be fully represented by its list of edges. Another common representation of a graph, which we explore in depth, is the connectivity matrix. Connectivity matrix M is a square matrix where each element $m(i, j) = 1\,(0)$ if there is (is not) an edge connecting the vertices i and j. For undirected graphs this matrix is symmetric. Figure 49.1 shows a simple graph and its associated connectivity matrix.



$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

Figure 49.1. On the right is a graph of six nodes numbered from top to bottom. On the left is its associated connectivity matrix.

© 2005 by Chapman & Hall/CRC

As a matter of convention, the connectivity matrix diagonal consists of either zeros or ones. Ones are frequently used based on the simple assertion that each vertex is connected to itself. We use the convention where the diagonal is filled with zeros. Justification is provided later.

A *walk* of length n is a set of edges expressed as an ordered list of n edges $((i_0, j_0),(i_1, j_1), \ldots, (i_n, j_n))$, where each vertex j_a is the same as vertex i_{a+1}. A *path* of length n is a walk where all i_a are unique. If j_n is the same as i_0, then the path forms a *cycle*.

A connected component is a set of vertices where there is a path between every two vertices in the component. The graph in Figure 49.1 has two connected components. In the case of di-graphs, this would be called a fully connected component. A complete graph has an edge directly connecting any two vertices in the graph. A complete subgraph is a subset of vertices in the graph with edges directly connecting any two members of the set.

We use the following property of connectivity matrices: element $m^k(i, j)$ of the power k of graph G's connectivity matrix M (i.e. M^k) is the number of walks of length k from vertex i to vertex j on G [11]. This can be verified using the definition of matrix multiplication and the definition of the connectivity matrix. The computation of each element of $m^k(i, j)$ checks for the existence of a two-hop path from i to j via any intermediate node.

You can compute successive powers k of M until $M^k = M^{k-1}$ to find the connected components of a graph. The resulting matrix will contain disjoint equivalence classes. Any zero element $m^k(i, j)$ in M^k signifies that elements i and j belong to different connected components (equivalence classes).

The diagonal of zero convention reduces the influence of cycles when computing powers of M. For example, we may look for the existence of a path of length 3 between nodes j and h. This should not include the walk (j, h), (h, j), (h, j). Maintaining a diagonal of zero in successive values of M^k does this.

We also present probabilistic connectivity matrices for random graph classes. These matrices replace the binary values for connectivity matrix elements $m(k, j)$ with the associated probabilities of an edge existing between nodes k and j. In which case, graph instances can be constructed by traversing the upper triangular half of the probabilistic connectivity matrix L and comparing the element value $l_{i,j}$ to a uniform random variable r in the range zero to one. If $l_{i,j} < r$, then an edge is inserted between nodes i and j. If not, none exists. A nonprobabilistic connectivity matrix is made by setting $l_{i,j}$ to one in the first case, else it is zero. For the Erdös–Rényi and small-world graphs, this method is essentially identical to current graph construction methods. For scale-free graphs, this technique is new.

49.4 Erdös–Rényi Graphs

The first model we discuss in detail is the Erdös–Rényi random graph [12]. These graphs are defined by the number of nodes n and a uniform probability p of an edge existing between any two nodes (Figure 49.2). Let us use E for $|E|$ the number of edges in the graph. Since the degree of a node is the result of multiple Bernoulli trials with probability p, the degree of an Erdös–Rényi random graph follows a Bernoulli distribution. As the graph scales, the number of nodes n approaches infinity, and the degree distribution converges asymptotically to a Poisson distribution.

The expected number of hops between any two nodes in this graph grows proportionally to the log of the number of nodes [13]. Note that Erdös–Rényi graphs do not necessarily form a single connected component. When $E - n/2 \ll -n^{2/3}$ the graph is subcritical and almost certainly not connected. A phase change occurs in the critical phase where $E = n/2 + O(n^{2/3})$, and in the supercritical phase, where $E - n/2 \gg -n^{2/3}$, a single giant component becomes almost certain. When $E = n \log n/2 + O_p(n)$ the graph is fully connected [14]. The expected number of edges for an Erdös–Rényi graph is $n(n-1)\,p\,/2$.

To construct a probabilistic connectivity matrix for this graph (Figure 49.3), create an n-by-n matrix with all elements on the diagonal set to zero and all the other elements set to p. If $n = 3$ and $p = 0.25$, then we get

$$\begin{bmatrix} 0 & 0.25 & 0.25 \\ 0.25 & 0 & 0.25 \\ 0.25 & 0.25 & 0 \end{bmatrix} \tag{49.1}$$

© 2005 by Chapman & Hall/CRC

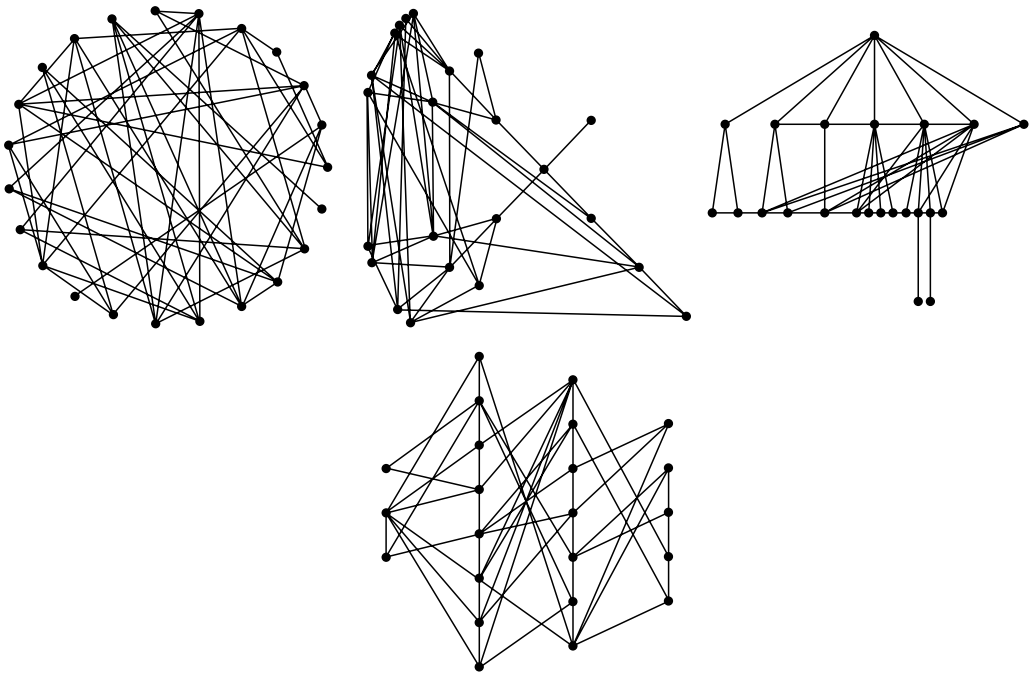


Figure 49.2. Example Erdös–Rényi graphs with $n = 23$ nodes and probability $p = 0.2$. Clockwise from upper left: nodes in a circle, radial embedding, ranked embedding by geodesic distance from three nodes chosen at random, and rooted embedding from a random node.

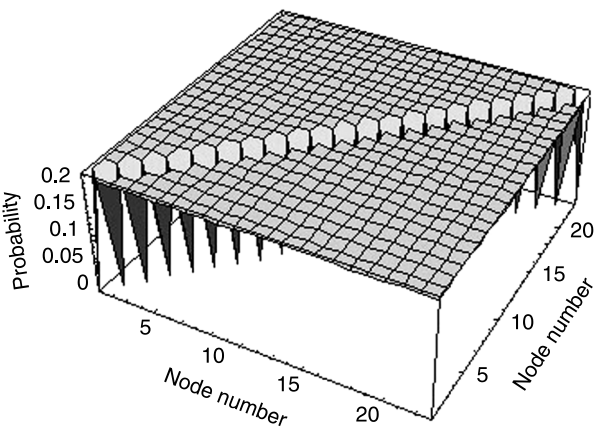


Figure 49.3. A three-dimensional plot of the probabilistic connectivity matrix for Erdös–Rényi graphs with $n = 23$ and $p = 0.2$. Diagonal values are zero. All other edges have the same probability.

Erdös–Rényi graphs are given for two reasons: (i) they are the best-known random graph class with many established properties; (ii) they are simple and have tutorial value.

49.5 Small-World Graphic

The second graph class considered is the family of small-world graphs introduced by Watts [2]. They have two characteristics: (i) a small expected value for the number of hops between any two nodes;

© 2005 by Chapman & Hall/CRC

(ii) significant clustering among the nodes. The rate of increase of the number of hops between nodes for these graphs is roughly equivalent to that of Erdös–Rényi graphs [2].

For small-world graphs example we use the *connected caveman model* [2] To construct connected caveman graphs, use this procedure:

- Construct a set of complete subgraphs.
- In each subgraph, one edge is removed and replaced with an edge connecting the subgraph to the next one, forming a cycle reminiscent of a "string of pearls."
- A fixed number of edges in the system are replaced with random edges.

Watts [2] also explores other small-world graph examples. The ideas presented here are also applicable to those examples. The connected caveman small-world model has three parameters: (i) n, the number of nodes in the graph; (ii) c, the number of nodes in each subgraph; (iii) e, the number of edges rewired.

The node degree distribution in this model is nearly constant. The mean node degree value is $c-1$, where c is the number of nodes in the complete subgraph. Variance around the mean is caused by two sets of Bernoulli trials: (i) the probability an edge connected to the vertex is chosen to create the "string of pearls," raising (or reducing) the node degree by one; (ii) the likelihood of an edge attached to the vertex being rewired.

This graph structure is discussed because: (i) the small expected number of hops between nodes is attractive for sensor network applications; (ii) it is a structure that can be easily maintained as a distributed system; (iii) it has a partially deterministic structure. The small-world connectivity matrix is constructed by essentially using the algorithm that creates a graph instance. For the connected caveman model example (Figure 49.4):

1. Create an $n \times n$ matrix
2. Populate the diagonal of the matrix with $c \times c$ blocks of value one. If n is not a multiple of c, then the last block will be $n \bmod c$ by $n \bmod c$. This matrix is block diagonal.
3. Set diagonal values to zero.
4. Connect the fully connected components. For all blocks: block starting address j and last address $k = j + c$. Set elements $(k-1, k)$ and $(k, k-1)$ to zero. Set elements $(k+1, k)$ and $(k, k+1)$ to zero. Set element $(n-1, n)$ and $(n, n-1)$ to zero, and set $(1, n)$ and $(n, 1)$ to one.
5. Count the zeros and ones in the matrix, excluding the diagonal. The probability of gaining a connection by the rewiring step becomes $2e/$(number of zeros). The probability of losing a connection by the rewiring step becomes $2e/$(number of ones).
6. For all elements of the matrix, except diagonals (which remain zero), if the element is one (zero) then subtract (add) the probability of losing (gaining) a connection.

The resulting matrix expresses the probabilities of edges existing in the connected caveman model. For other examples, similar algorithms can easily be constructed. For our example, the matrix for $n = 6$, $c = 3$, and $e = 1$ is

$$\begin{bmatrix} 0 & \frac{5}{6} & \frac{5}{6} & \frac{1}{9} & \frac{1}{9} & \frac{5}{6} \\ \frac{5}{6} & 0 & \frac{1}{9} & \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{5}{6} & \frac{1}{9} & 0 & \frac{5}{6} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{5}{6} & 0 & \frac{5}{6} & \frac{5}{6} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} & \frac{5}{6} & 0 & \frac{1}{9} \\ \frac{5}{6} & \frac{1}{9} & \frac{1}{9} & \frac{5}{6} & \frac{1}{9} & 0 \end{bmatrix} \tag{49.2}$$

Figure 49.5 shows an example graph generated using the probabilities in the matrices in Equations (49.2) and (49.3).

It is possible to modify step 4 of the probabilistic connectivity matrix generation procedure to choose block elements at random. Among other things, the matrix created in this manner is regular in

© 2005 by Chapman & Hall/CRC

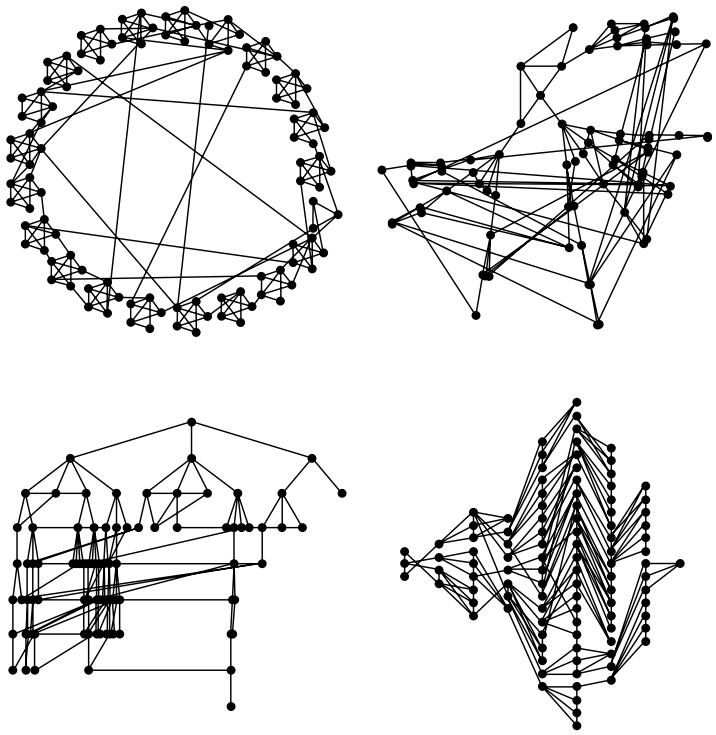


Figure 49.4. Example connected caveman graphs with 103 nodes n starting from connected subgraphs of $c = 5$ members each. A total of 22 edges (e) were rewired at random. Clockwise from upper left: "string of pearls," ranked embedding by geodesic distance from nodes 101, 102 and 103, radial embedding, and rooted embedding from node 103.

a certain sense. We call graphs with this property rpm-graphs (regular probability matrix). Perform the procedure above but omit step 4. At the end, modify the matrix as follows:

- All edges connecting components in the cluster have the same probability of being chosen, $1/[c(c-1)]$.
- This value is subtracted from nondiagonal elements in the block representing the cluster on the diagonal.
- Each node in the current cluster has the same probability of being selected ($1/c$) for connection to the next cluster. Each node in the next cluster also is equally likely ($1/c$) to receive an edge.
- To represent this, each element potentially connecting the two clusters has a probability of $1/c^2$ added to it.

Note that when n is not 0 modulo c, the block size of the final subgraph is not c but n modulo c. For the final block, the values above become $1/[c'(c'-1)]$ and $1/(c'c)$, where $c' = n$ modulo c. Notice that, since each row is a permutation of the same probabilities, both this description and the Erdös–Rényi description are regular. The regular matrix description for $n = 6$, $c = 3$, and $e = 1$ is

$$\begin{bmatrix} 0 & \frac{2}{3} & \frac{2}{3} & \frac{2}{9} & \frac{2}{9} & \frac{2}{9} \\ \frac{2}{3} & 0 & \frac{2}{3} & \frac{2}{9} & \frac{2}{9} & \frac{2}{9} \\ \frac{2}{3} & \frac{2}{3} & 0 & \frac{2}{9} & \frac{2}{9} & \frac{2}{9} \\ \frac{2}{9} & \frac{2}{9} & \frac{2}{9} & 0 & \frac{2}{3} & \frac{2}{3} \\ \frac{2}{9} & \frac{2}{9} & \frac{2}{9} & \frac{2}{3} & 0 & \frac{2}{3} \\ \frac{2}{9} & \frac{2}{9} & \frac{2}{9} & \frac{2}{3} & \frac{2}{3} & 0 \end{bmatrix} \tag{49.3}$$

© 2005 by Chapman & Hall/CRC

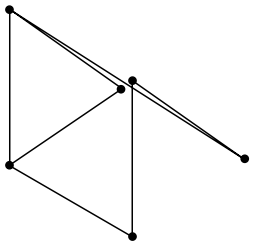


Figure 49.5. Connected caveman graph with $n = 6$, $c = 3$, and $e = 1$.

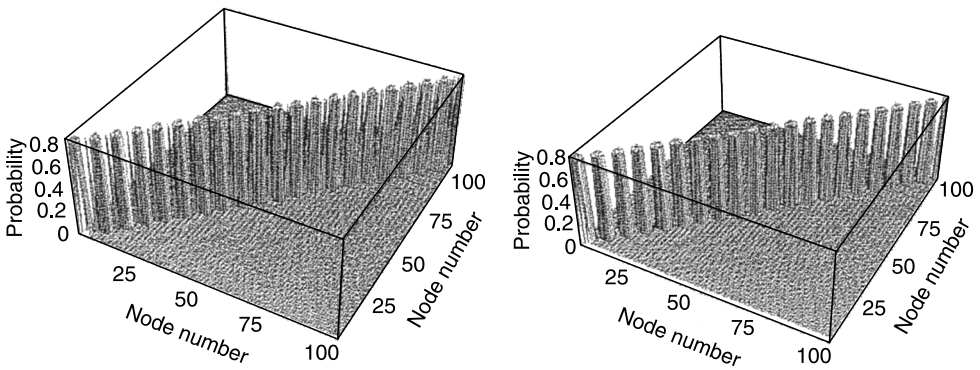


Figure 49.6. Three-dimensional plot of matrices for connected caveman model with $n = 103$, $c = 5$, and $e = 22$. Left: first method given. Right: regular matrix. Note how clear the clustering is and low the probability of other connections is.

Note that the description of rpm-graphs is regular because each node has an equivalent probability density function and each row of the connectivity matrix is a permutation of every other row. The degree of each node thus has the same expected value. This does not mean that all nodes in a given instance of the rpm-graph will have the same degree. A comparison of the two methods for generating a matrix using the connected caveman model is given in Figure 49.6.

49.6 Scale-Free Graphs

The third graph class is the scale-free model. It comes from empirical analysis of real-world systems, such as e-mail traffic, the World Wide Web, and disease propagation [13]. In scale-free graphs the node degree distribution varies as an inverse power law (i.e. $P[d] \propto d^{-\gamma}$). They are called scale free because the power law structure implies nodes existing with nonzero probability at all possible scales. The expected number of hops for scale-free networks is smaller than the expected number of hops for Erdös–Rényi and small-world graphs [13]. Scale-free graphs are defined by two parameters: the number of nodes n and the scaling factor γ. Empirical analyses done by different research groups at different times find the Internet's γ parameter value ranging from 2.1 to 2.5 [13]. Of the random graph classes discussed, node degree variance in this class is the largest.

Scale-free networks are discussed for several reasons: (i) they appear in existing systems like the Internet and are of use for large-scale wired networks; (ii) the small number of expected hops between nodes is attractive; (iii) studies indicate that their structure has unique dependability properties [3]; (iv) epidemiological studies using this model indicate that there may be parallels between biological

© 2005 by Chapman & Hall/CRC

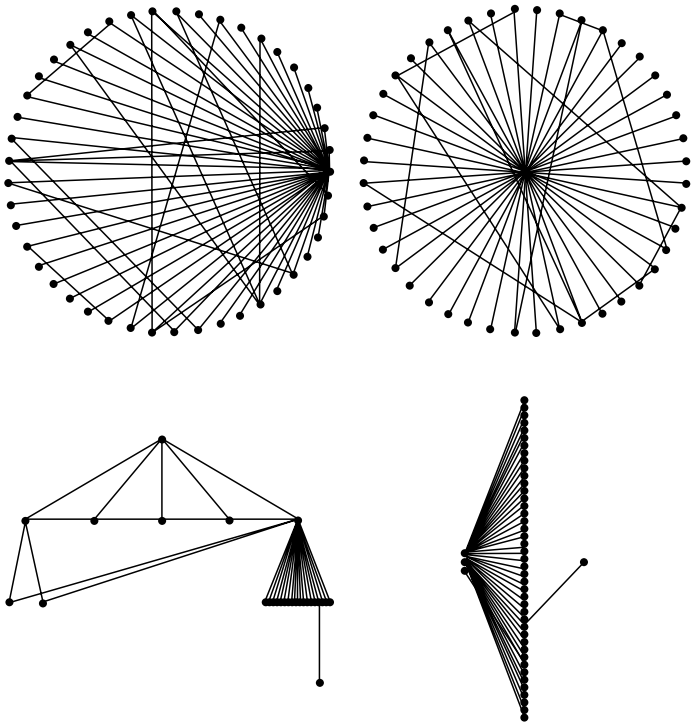


Figure 49.7. Example scale-free graphs with $n = 45$ and $\gamma = 3.0$. Clockwise from top left: nodes in a circle, radial embedding, ranked embedding in order of the geodesic distance from the three largest hubs, and rooted embedding with the root set as the second largest hub.

pathogen propagation and computer virus propagation [4]. An algorithm constructing these graphs using positive feedback and producing graphs with $\gamma \approx 3$ can be found in Barabási and Albert [15]. Barabási and Albert's use of positive feedback plausibly explains how scale-free systems emerge and why they are widespread.

Figure 49.7 illustrates how scale-free graphs differ from small-world and Erdös–Rényi graphs. The majority of nodes have degree one or two, but there exists a small number of hub nodes with a very large degree. Erdös–Rényi and small-world graphs have an almost flat architecture with node degree clustered about the mean. The hub nodes dominate the topology of the scale-free graphs. The ranked embedding illustrates that it is extremely unlikely that a node would be many hops away from a major hub.

Creating a probabilistic connectivity matrix for scale-free graphs is challenging. As noted above, scale-free graphs are characterized by n the number of nodes and γ the scaling factor. The first step is to compute the probability distribution for node degree d. Remember $P[d] \propto d^{-\gamma}$. We compute the probability distribution by finding a constant factor for all probabilities to sum to unity. Set

$$P[d] = bd^{-\gamma} \tag{49.4}$$

Since node degree ranges from 1 to $n-1$:

$$1 = \sum_{d=1}^{n-1} bd^{-\gamma} \tag{49.5}$$

© 2005 by Chapman & Hall/CRC

thus

$$b = \frac{1}{\sum_{d=1}^{n-1} d^{-\gamma}} \tag{49.6}$$

We now have a closed-form solution for the node degree probability distribution.

The next step is to determine how many edges are incident on each node. Construct a vector v of $n-1$ elements, whose values range from zero to one. Each element k of the vector contains the value

$$v[k] = \sum_{d=1}^{k-1} bd^{-\gamma} \tag{49.7}$$

Vector element $v[0]$ has the value zero and element $v[n-1]$ has the value one. Each element represents the probability of a node existing of degree less than or equal to k.

Each row of the probabilistic connectivity matrix represents the expected behavior of $1/n$th of the nodes of the class under consideration. We now construct a vector v' of n elements; the value of $v'[k]$ states how many edges are incident on node k. Set $v'[k]$ to the index of the largest element of v whose value is less than or equal to k/n.[1]

The elements of the connectivity matrix are probabilities of connections between individual nodes. These values are computed using the insight from Barabási and Albert [15], that scale-free networks result from positive feedback. Nodes are more likely to connect to other nodes with many connections. The value of each matrix element (k, i) is therefore

$$P[k,i] = \frac{v''[i]v'[k]}{\sum_{m \neq k} v'[m]} \tag{49.8}$$

The likelihood of choosing another node i to receive a given edge from the current node k is the degree of i divided by the sum of the degree of all nodes except k. Summing these factors would give a total probability of one for the row. Since k has degree $v'[k]$ these probabilities are multiplied by $v'[k]$, so that the total of the probabilities for the row is k. This finishes the derivation of Equation (49.8).

We modify the values of Equation (49.8) in two ways. Since node degrees have an exponential distribution, the values of the bottom rows are often much larger than the other degrees. The result of Equation (49.8) for values of k and l close to n can be greater than one. To avoid having elements of the matrix with values greater than one (i.e. probability greater than one), we compute the matrix elements in a double loop starting with k (outer loop) and i (inner loop) set to $n-1$. The values of k and i are decremented from $n-1$ to zero. If the value of Equation (49.8) is greater than one then the corresponding element is set to one and the value copied from $v'[k]$ for computing row k is decremented by one. This keeps all matrix elements in the range zero to one, so that they represent probabilities.

The other modification of element values that deviates from Equation (49.8) forces the matrix to be symmetric. When computing a row k and $k < n-1$, all elements for $i > k$ are set to be the same as the values computed for element (i, k). If the value of element (i, k) is one, then the value copied from $v'[k]$ is again decremented. In some cases this may force the sum of row k to deviate from $v'[k]$. If the deviation is significant enough, then the resulting connectivity matrix may only have a degree

[1]It would also be possible to use the mean, or a weighted average, of the index values that point to elements in the range $(k-1)/n$ to k/n. Since Equation (49.8) can give values greater than one, constraining matrix element values to the range [0..1] flattens the degree distribution. Using the maximum index value counteracts this tendency.

© 2005 by Chapman & Hall/CRC

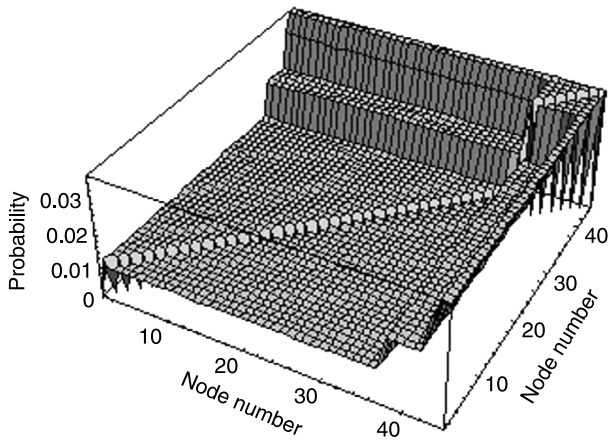


Figure 49.8. Three-dimensional plot of the connectivity matrix for a scale-free graph with $n = 45$ and $\gamma = 3.0$. Note the zero diagonal and the high probability of connections to the hub nodes. Connections between hub nodes are virtually assured. Connections between nonhub nodes are very improbable.

distribution that approximates the scaling factor γ. An example connectivity matrix for $n = 10$ and $\gamma = 2.0$ is

$$\begin{bmatrix}
0 & \frac{1}{22} & \frac{1}{22} & \frac{1}{22} & \frac{1}{22} & \frac{1}{22} & \frac{1}{10} & \frac{1}{10} & \frac{2}{9} & \frac{9}{10} \\
\frac{1}{22} & 0 & \frac{1}{22} & \frac{1}{22} & \frac{1}{22} & \frac{1}{22} & \frac{1}{10} & \frac{1}{10} & \frac{2}{9} & \frac{9}{10} \\
\frac{1}{22} & \frac{1}{22} & 0 & \frac{1}{22} & \frac{1}{22} & \frac{1}{22} & \frac{1}{10} & \frac{1}{10} & \frac{2}{9} & \frac{9}{10} \\
\frac{1}{22} & \frac{1}{22} & \frac{1}{22} & 0 & \frac{1}{22} & \frac{1}{22} & \frac{1}{10} & \frac{1}{10} & \frac{2}{9} & \frac{9}{10} \\
\frac{1}{22} & \frac{1}{22} & \frac{1}{22} & \frac{1}{22} & 0 & \frac{1}{22} & \frac{1}{10} & \frac{1}{10} & \frac{2}{9} & \frac{9}{10} \\
\frac{1}{22} & \frac{1}{22} & \frac{1}{22} & \frac{1}{22} & \frac{1}{22} & 0 & \frac{1}{10} & \frac{1}{10} & \frac{2}{9} & \frac{9}{10} \\
\frac{1}{10} & \frac{1}{10} & \frac{1}{10} & \frac{1}{10} & \frac{1}{10} & \frac{1}{10} & 0 & \frac{1}{5} & \frac{4}{9} & 1 \\
\frac{1}{10} & \frac{1}{10} & \frac{1}{10} & \frac{1}{10} & \frac{1}{10} & \frac{1}{10} & \frac{1}{5} & 0 & \frac{4}{9} & 1 \\
\frac{2}{9} & \frac{2}{9} & \frac{2}{9} & \frac{2}{9} & \frac{2}{9} & \frac{2}{9} & \frac{4}{9} & \frac{4}{9} & 0 & 1 \\
\frac{9}{10} & \frac{9}{10} & \frac{9}{10} & \frac{9}{10} & \frac{9}{10} & \frac{9}{10} & 1 & 1 & 1 & 0
\end{bmatrix} \tag{49.9}$$

Figure 49.8 shows a three-dimensional plot of the connectivity matrix for a scale-free graph with $n = 45$ and $\gamma = 3.0$.

49.7 Percolation Theory

Percolation theory is a study of flow through random media. Commonly, the random media are modeled as regular tessellations of a d-dimensional space where vertices are points connecting edges. It is also possible to consider arbitrary graphs. Two different models exist: site percolation is where vertices are either occupied or empty and bond percolation is where edges are either occupied or empty. We discuss only bond percolation; however, note that it is possible to create dual graphs to convert site (bond) percolation problems to bond (site) problems. These graphs can be considered models of wireless networks with a cellular infrastructure.

As an example, we construct probabilistic transition matrices for a rectangular tessellation of a two-dimensional space. This model requires three parameters: x the number of nodes in a row, y the number of nodes in a column, and p the probability of an edge being occupied. Note that n, the total number of

© 2005 by Chapman & Hall/CRC

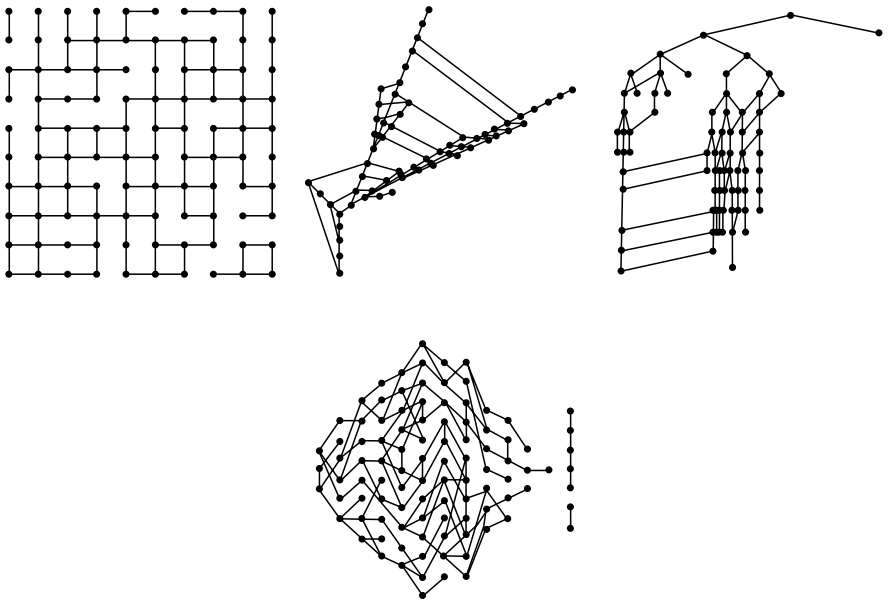


Figure 49.9. Different embeddings of a regular 10×10 matrix. Edge probability was set at 0.75. Top row from left to right: grid, radial embedding, and rooted embedding with node 50 as the root. Bottom row: ranked embedding from nodes 38, 39, and 40.

nodes, is equal to xy. The matrix construction method is only valid for finite problems. Once the matrix has been constructed, however, scaling analysis can be performed to consider infinite ranges. Figure 49.9 shows an example graph.

Excluding edge effects in this tessellation, each node (i, j) has four immediate neighbors: $(i+1, j)$, $(i, j+1)$, $(i-1, j)$, and $(i, j-1)$. Each vertex is assigned the unique row position $(i+(j-1)y)$ in the connectivity matrix. (This assumes that i(or j) ranges from zero to x (or y) and makes the matrix row major. Readers who are dogmatic about C or FORTRAN can change these conventions at will [16].) Vertices outside the range $([1..x], [1..y])$ are ignored, since they are out of bounds. All positions are set to zero in the connectivity matrix, except that for each node (i, j) the positions in its row $(i+(j-1)y)$ that correspond to its neighbors

$$i-1+jy \quad i+1+jy \quad i+(j-1)y \quad i+(j+1)y \tag{49.10}$$

are set to p. The matrix corresponding to a 3×3 grid with probability of 0.75 is

$$\begin{bmatrix} 0 & 0.75 & 0 & 0.75 & 0 & 0 & 0 & 0 & 0 \\ 0.75 & 0 & 0.75 & 0 & 0.75 & 0 & 0 & 0 & 0 \\ 0 & 0.75 & 0 & 0 & 0 & 0.75 & 0 & 0 & 0 \\ 0.75 & 0 & 0 & 0 & 0.75 & 0 & 0.75 & 0 & 0 \\ 0 & 0.75 & 0 & 0.75 & 0 & 0.75 & 0 & 0.75 & 0 \\ 0 & 0 & 0.75 & 0 & 0.75 & 0 & 0 & 0 & 0.75 \\ 0 & 0 & 0 & 0.75 & 0 & 0 & 0 & 0.75 & 0 \\ 0 & 0 & 0 & 0 & 0.75 & 0 & 0.75 & 0 & 0.75 \\ 0 & 0 & 0 & 0 & 0 & 0.75 & 0 & 0.75 & 0 \end{bmatrix} \tag{49.11}$$

Different embeddings of a regular 10×10 matrix and a three-dimensional plot of the connecting matrix for a 10×10 grid are shown in Figure 49.9 and Figure 49.10 respectively.

© 2005 by Chapman & Hall/CRC

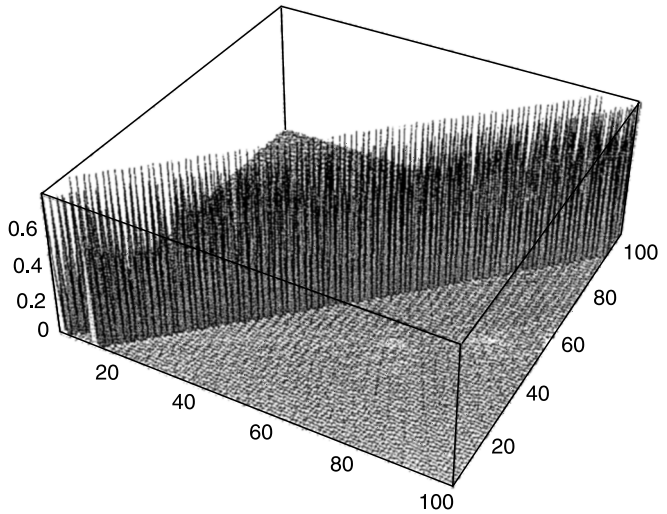


Figure 49.10. Three-dimensional plot of the connectivity matrix for a 10×10 grid with $n = 0.75$. It is a band-diagonal matrix.

49.8 *Ad Hoc* Wireless

Scale-free networks provide good statistical descriptions of large, evolving, wired networks with no centralized control. Wireless networks are also of importance. In particular *ad hoc* wireless networks, which have no fixed infrastructure, are suited to analysis as a type of random graph. Krishnamachari *et al.* [6] explain a fixed radius model for random graphs used to analyze phase-change problems in *ad hoc* network design.

The model of Krishnamachari *et al.* [6] places nodes at random in a limited two-dimensional region. Two uniform random variables provide a node's x and y coordinates. Two nodes in proximity to each other have a very high probability of being able to communicate. For this reason, they calculate the distance r between all pairs of nodes. If r is less than a given threshold, then an edge exists between the two nodes. In their work, many similarities are found between this graph class and the graphs studied by Erdös and Rényi. Their analysis looks at finding phase transitions for constraint satisfaction problems. These graphs differ from Erdös–Rényi graphs in that they have significant clustering, like the small-world graph class. We will use the Krishnamachari *et al.* [6], model except that, where they create an edge with probability one when the distance between two nodes is less than the threshold value, we will allow the probability to be set to any value in the range [0..1]. Figure 49.11 shows an example range-limited random graph.

We construct range-limited graphs from the following parameters:

- n, the number of nodes
- max_x (max_y), the size of the region in the x (y) direction
- r, the maximum distance between nodes where connections are possible
- p, probability that an edge exists connecting two nodes within the range

Construction of range-limited random graphs proceeds in two steps: (i) sort the nodes by either their x (or possibly y) coordinate and use order statistics to find the expected values of that coordinate; (ii) determine probabilities for edges existing between nodes based on these expected values.

To construct the connectivity matrix for range-limited graphs, we consider the position of each node as a point defined by two random variables, i.e. the x and y locations. Without loss of generality, we use normalized values for the x, y, and r variables limiting their range to [0..1]. To calculate probabilities,

© 2005 by Chapman & Hall/CRC

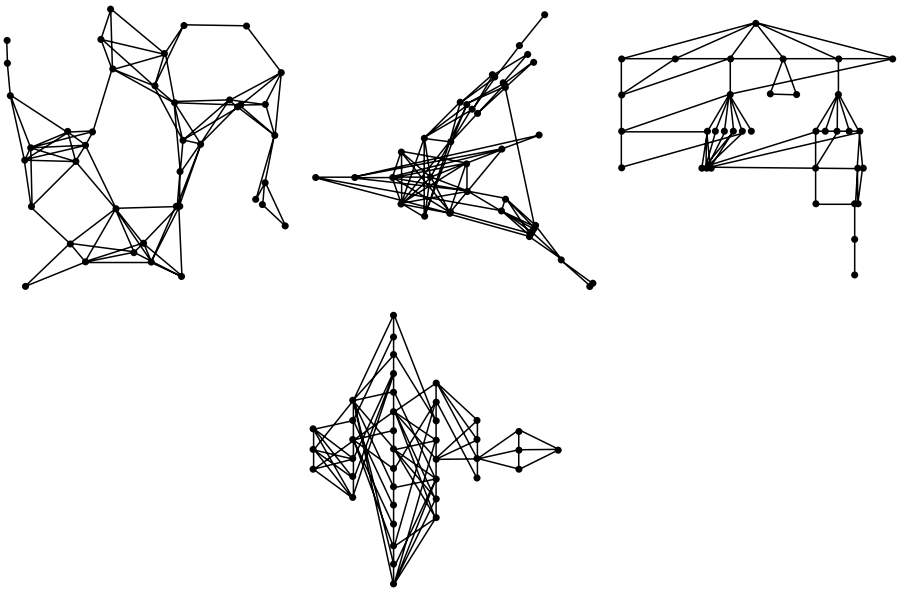


Figure 49.11. Different embeddings of a range-limited random graph of 40 nodes positioned at random in a unit square region. The distance threshold was set as 0.25, and within that range the edges exist with a probability of one. Clockwise from upper left: geographic locations, radial embedding, ranked embedding from nodes 38, 39, and 40, and rooted embedding with node 40 as the root.

we sort each point by its x variable. For the n nodes, rank statistics provide expected value $j/(n+1)$ for the node in position j in the sorted list. Using Euclidean distance, an edge exists between two nodes j and k with probability p when

$$(x_j - x_k)^2 + (y_j - y_k)^2 \leq r^2 \tag{49.12}$$

By entering the expected values for nodes of rank j and k and reordering terms, this becomes

$$(y_j - y_k)^2 \leq r^2 - \left(\frac{j}{n+1} - \frac{k}{n+1}\right)^2 \tag{49.13}$$

We assume that the random variables giving the x and y positions are uniformly distributed and uncorrelated. The probability of this occurring is the probability that the square of the difference of two normalized uniform random variables is less than the constant value c provided by the right-hand side of Equation (49.13). Two uniform random variables describe a square region, where every point is equally likely. Equation (49.13) is an inequality, so it defines a closed linear region. Because the right-hand side is squared, two symmetric regions are excluded from the probability. The limiting points are when y_j or y_k are equal to the constant on the left-hand side of Equation (49.13). Algebraic manipulation provides the equation $2c - c^2$ for the probability of Equation (49.13) occurring. An example matrix for six nodes in a unit square with $r = 0.3$ and $p = 1.0$ is

$$\begin{bmatrix} 0 & 0.134 & 0.0167 & 0 & 0 & 0 \\ 0.134 & 0 & 0.134 & 0.0167 & 0 & 0 \\ 0.0167 & 0.134 & 0 & 0.134 & 0.0167 & 0 \\ 0 & 0.0167 & 0.134 & 0 & 0.134 & 0.0167 \\ 0 & 0 & 0.0167 & 0.134 & 0 & 0.134 \\ 0 & 0 & 0 & 0.0167 & 0.134 & 0 \end{bmatrix} \tag{49.14}$$

© 2005 by Chapman & Hall/CRC

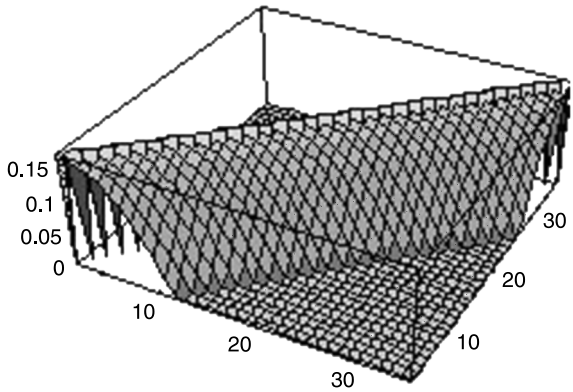


Figure 49.12. Three-dimensional plot of the connectivity matrix for a range-limited graph of 35 nodes with range of 0.3.

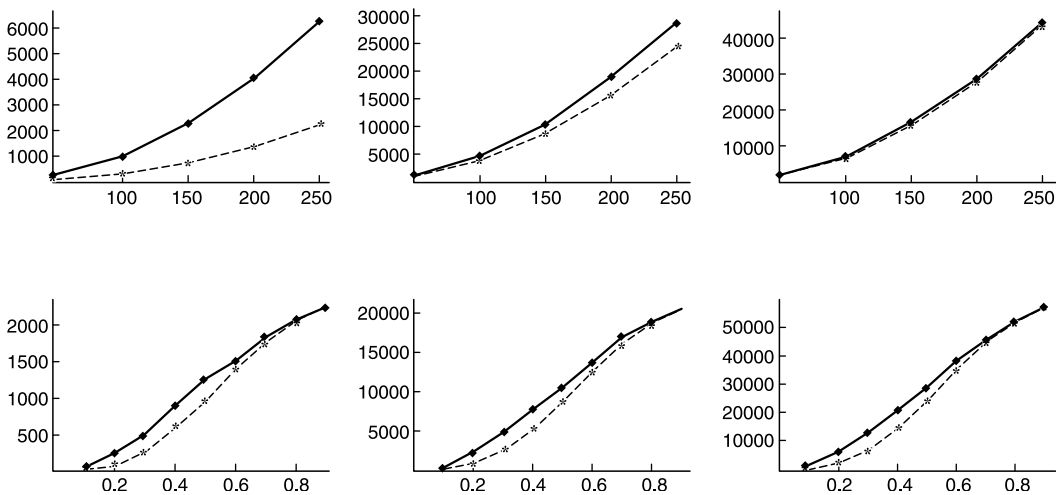


Figure 49.13. Number of edges (y axis) as a function of n and r (x axis). The top row varies n from 50 to 250 with r fixed. From left to right: $r = 0.2$, 0.5, and 0.7. Bottom row varies r from 0.1 to 0.9 while keeping n fixed. From left to right: 50, 150, and 250. For both rows of graphs the y-axis is the mean number of edges and the x-axis is the variable being varied. Five repetitions were made of each data point. The 90% confidence intervals are shown.

Figure 49.12 shows a three-dimensional plot of an example matrix. Figure 49.13 compares the number of edges for range-limited graphs constructed directly versus those constructed using the probabilistic connectivity matrices as a function of n and r. The approximation achieved by this approach is good, but far from perfect.

49.9 Cluster Coefficient

The clustering coefficient expresses the cliquishness of the network ($C = 1$ for a complete graph, $C = 0$ for a tree). It is the percentage of nodes two hops away that are also only one hop away. This can also be expressed as the percentage case of nodes reached in two hops that were already reached in one hop, or the likelihood that a friend's friend is also a friend of mine. It is defined as [17] (the two definitions

© 2005 by Chapman & Hall/CRC

are equivalent)

$$C = \frac{3 \times (\text{Number of triangles in network})}{\text{Number of connected triples of verties}}$$

$$C = \frac{6 \times (\text{Number of triangles in network})}{\text{Number of paths of length two}} \tag{49.15}$$

We developed a new algorithm for computing C based on the fact that each element $a_{i,j}$ of a graph's connectivity matrix raised to the power k contains the number of paths of length k from i to j. We square the connectivity matrix M to compute M^2. We consider all nondiagonal elements of M and M^2 and compute two sums: the elements of M^2 and the elements of M^2 where the corresponding values of M are not zero. The second sum is the numerator and the first sum is the denominator of the cluster coefficient.

Watts [2] gives a different definition for cluster coefficient: the number of edges between nodes in the subgraph of nodes immediately adjacent to a given node are divided by the total number of edges possible given the degree of the node. The sum of these values is averaged. This definition has been deprecated [18] in favor of the definitions of Newman [17]. The above definition is equivalent to computing this definition, except that the ratio is over the entire graph instead of at each node. The approach above provides a more stable answer. The deprecated definition is

$$C = \frac{\text{Number of edges between neighbors of the node}}{\text{Number of possible links in local neighborhood}}$$

$$C = \text{Average}\left[\frac{\text{Number of edges between neighbors of the node}}{d(d-1)/2}\right] \tag{49.16}$$

How the cluster coefficient C relates to the characteristics of random graph classes is shown in Figures 49.14 and 49.15.

For small-world graphs (see Figures 49.16–49.18), we notice the same relationship between cluster coefficient and graph-size growth independent of both parameter e and cluster size. Initially, the graph is dominated by a single fully connected cluster, giving it a high cluster coefficient. This value falls off rapidly once multiple clusters exist in the graph. As the number of clusters increases, so the cluster coefficient asymptotically approaches a value close to the inital value. This is valid for the connected caveman model from Watts [2]. We start our calculations with graph sizes larger than the cluster size. When the graph size and the number if random rewirings are held fixed, the cluster coefficient is

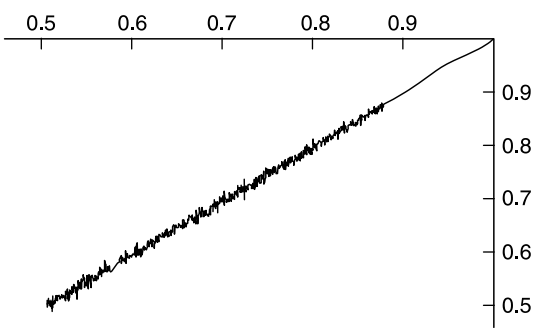


Figure 49.14. Cluster coefficient (y axis) versus edge probability (x axis) for Erdös–Rényi random graphs of 100 nodes. Results are similar for other graph sizes. As the graph size increases, the plot converges to a smooth curve.

© 2005 by Chapman & Hall/CRC

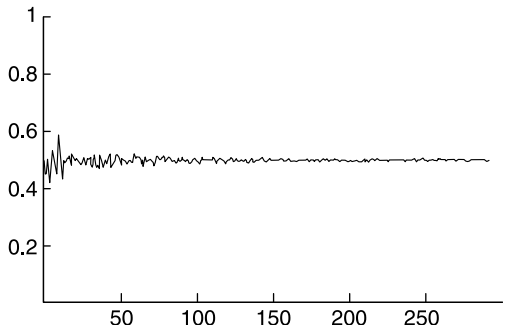


Figure 49.15. Graph of cluster coefficient (y axis) versus number of nodes in the graph (x axis) for Erdös–Rényi graphs, indicating that graph size does not significantly affect clustering coefficient mean value. The variance is greater at small node sizes. This effect is observable for most edge probabilities. As we have seen the edge probability strongly influences the cluster coefficient.

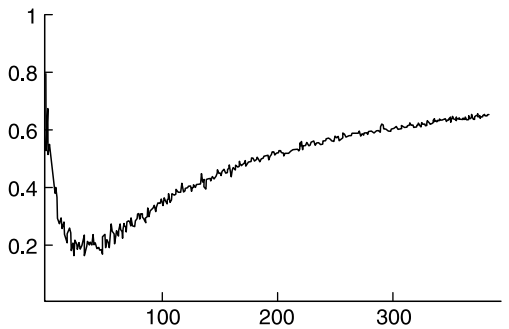


Figure 49.16. Cluster coefficient C (y axis) versus graph size (x axis) with fixed cluster size (seven) and number of rewirings (100).

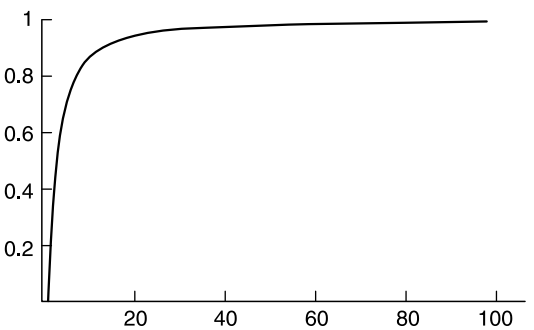


Figure 49.17. C (y axis) versus cluster size (x axis) for small-world graphs of 200 nodes and 100 edges rewired.

sensitive to the cluster size. Initially, the cluster coefficient increases the value dramatically then asymptotically approaches one. As the number of randomly rewired nodes increases, so the cluster coefficient decreases slowly.

For scale-free graphs, increasing the graph size causes the cluster coefficient to decline asymptotically. The shape of the curve in Figure 49.19 leads us to believe that the Barabási algorithm for creating scale-free graphs is biased to create clusters in the initial phase of graph creation. We suspect this is an artifact

© 2005 by Chapman & Hall/CRC

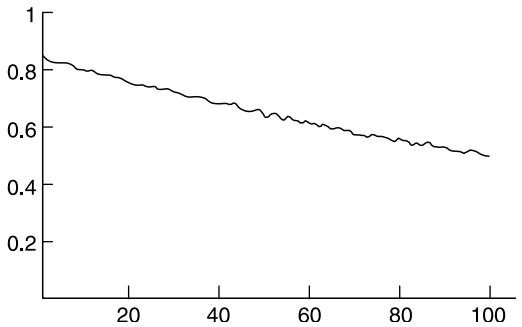


Figure 49.18. *C* (*y* axis) versus number of rewired nodes (*x* axis) for small-world graphs of 200 nodes and cluster size of seven.

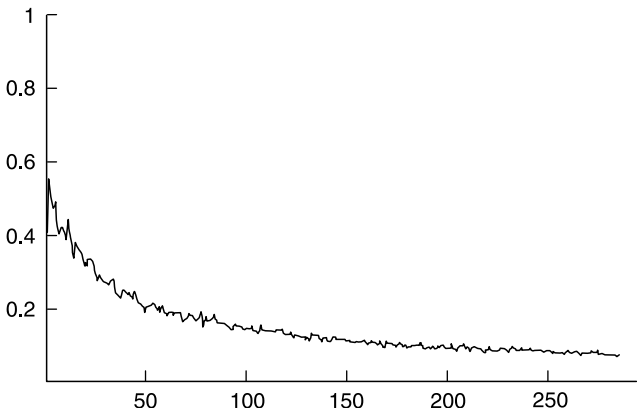


Figure 49.19. *C* (*y* axis) versus graph size (*x* axis) for scale-free graphs created using the Barabási algorithm with average node degree of five.

of the algorithm, rather than a property of scale-free graphs. On the other hand, the average node degree appears to influence the cluster coefficient significantly. Figure 49.20 shows the cluster coefficient versus average node degree for scale-free graphs of size 100 (top) and 250 (bottom).

49.10 Mutuality

Newman [17] introduces a mutuality factor Mu (*M* in Newman [17]) for the mean number of paths of length two leading to nodes two hops away:

$$\mathrm{Mu} = \frac{\mathrm{mean[number_of_vertices_2_hops_away]}}{\mathrm{mean[paths_of_length_2_to_those_vertices]}} = \frac{\mathrm{Average}\left\{d/\left[1 + C^2(d-1)\right]\right\}}{d_{\mathrm{average}}} \tag{49.17}$$

The estimator for z_2, the expected number of nodes first reachable in two hops, derived by Newman [17] is

$$z_2 = \mathrm{Mu}(1 - C)(\mathrm{Average}[d^2] - \mathrm{Average}[d]) \tag{49.18}$$

© 2005 by Chapman & Hall/CRC

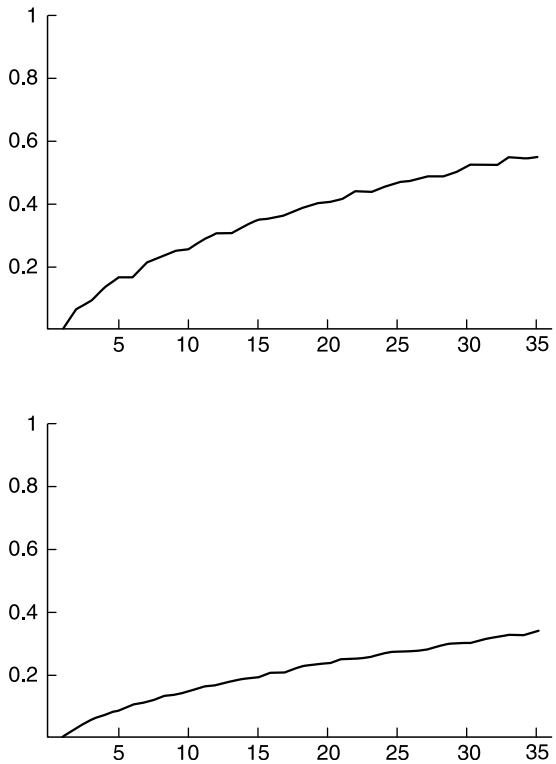


Figure 49.20. C (y axis) versus average node degree (x axis) for scale-free graphs created using the Barabási algorithm of 100 nodes (top) and 200 nodes (bottom).

Like C when considering the number of triangles in the network, Mu looks at the number of rectangles. For a random graph

$$c = (\text{Average}[d^2] - \text{Average}[d])\frac{\text{Mu}}{n(n-1)/2} \tag{49.19}$$

To analyze these factors, 35 graphs in each random graph class were created. The Erdös–Rényi graphs had 100 nodes with a uniform probability of 0.04 for an edge existing between two nodes. The small-world graphs had 100 nodes, connected components of four nodes, and had 11 edges rewired at random. The scale-free graphs had 100 nodes and the average node degree set to 4. Note that all the graphs were the same size and with the same average node degree.

For Erdös–Rényi graphs, the mean value of C was 0.0423888 with variance 0.000164686. For the small-world graphs, the mean value of C was 0.374865 and the variance was 0.000322814. For scale-free graphs, the mean value of C was 0.127691 and the variance was 0.0000785896.

Mu accounts for the overcount of nodes two hops away when using C to estimate the number of nodes reachable by a given number of hops. We extend the mutuality concept to nodes three or more hops away and refer to this as Mu[h]. The question is whether or not values of Mu[h] deviate significantly from unity. Our analysis indicates that it does.

For Erdös–Rényi graphs, average Mu[h] factors are plotted at the top Figure 49.21. The error bars are $\pm 2\sigma/\sqrt{35}$. The bottom figure shows the inverse of the mean. We see that for between two to nine hops the Mu factor is very significant. For five hops, five separate paths exist to each node on average.

© 2005 by Chapman & Hall/CRC

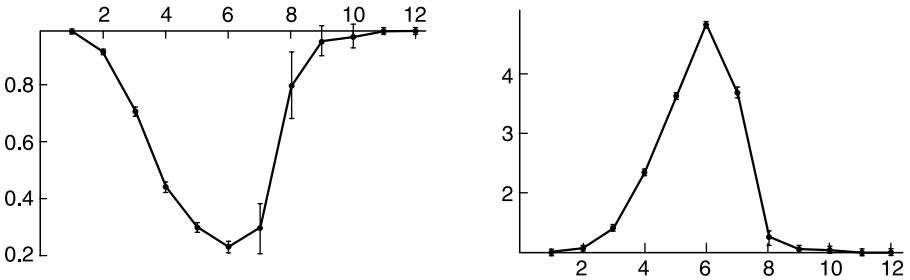


Figure 49.21. Right: mean values of Mu[h] (y axis) versus h (x axis) for Erdös–Rényi graphs. Left: inverse of the right-hand graph.

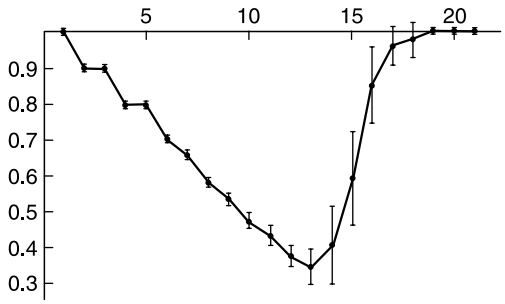


Figure 49.22. Mean values of Mu[h] (y axis) versus h (x axis) for small-world graphs.

The small error bars support the supposition that these values are likely to be consistent for the entire class of Erdös–Rényi graphs with these parameters.

It is interesting to note that the variance grows significantly after the value of Mu[h] has peaked. This implies that the number of different paths is fairly uniform as the number of independent paths increases. When the number of paths starts to decrease, the variance increases noticeably. The inference is that there are fewer paths, since most nodes have already been reached. We see later that the number of nodes reached by specific numbers of hops supports this inference. The number of paths to the remaining nodes becomes dependent on details of the graph structure of the particular instance and harder to predict.

For the set of small-world graphs, the shape of the Mu[h] function shown in Figure 49.22 is similar to the Erdös–Rényi Mu[h] function. Note that the number of independent paths grows more slowly. This is tied to the relatively small number of edges that have been rewired.

As shown in Figure 49.23, Mu[h] is most striking for scale-free graphs. The shape of the Mu[h] function is the same as for Erdös–Rényi and small-world graphs. The number of independent paths grows more quickly than for either of the other random graph models. It is interesting to note that the variance is almost undetectable. These graphs were constructed using Barabási's algorithm, where $\gamma \approx 3$. The results in Figure 49.23 are consistent with those of Reittu and Norros [19], where the expected number of hops when $2 < \gamma < 3$ scale as $\log \log n$.

These results show that the Mu[h] factor is even more significant when $h > 2$. Aspects of this factor appear to be common to all three models. Figure 49.24 shows Mu[h] for all three graph types.

We consider how Mu[h] varies for the different graph classes by plotting Mu[h] as parameters vary. For Erdös–Rényi graphs, the shape of the Mu[h] curve is sensitive to average node degree, which is a product of the number of nodes in the graph times the probability of an edge between two random nodes. For graphs with more than 100 nodes and edge probability over 20%, the shape in Figure 49.25

© 2005 by Chapman & Hall/CRC

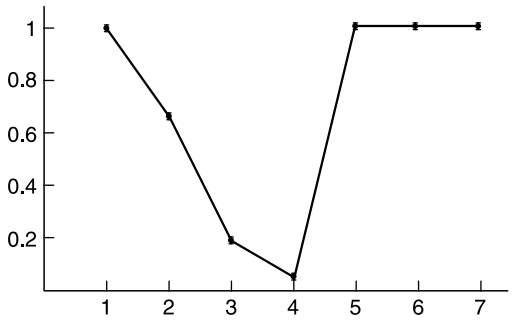


Figure 49.23. Mean values of Mu[h] (y axis) versus h (x axis) for scale-free graphs.

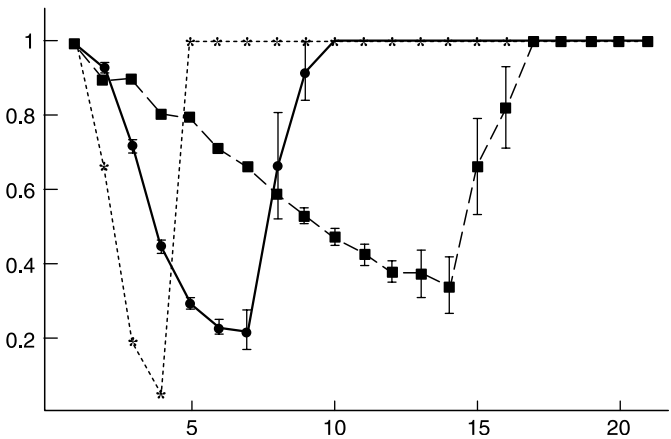


Figure 49.24. Mean values of Mu[h] (y axis) versus h (x axis) for the three graph types. From left: scale-free, Erdös–Rényi, and small world.

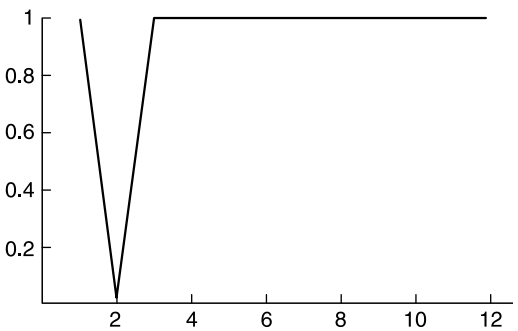


Figure 49.25. Mu[h] (y axis) versus h for an Erdös–Rényi graph of over 100 nodes and p over 20%.

with a large number of redundant paths of length two and no other redundant paths is typical. With graphs of 50 nodes, the same shape is typical with edge probability of 40% or higher. The shapes in Figure 49.26 occur for graphs of 100 and 130 nodes and probability of 10%. As the graph size increases, the shapes converge to the shape in Figure 49.25, with a spike at 2.

Figure 49.27 shows plots when 50 node graphs were generated. When the probability was less than 4% the plots tended to be flat lines, indicating that the graphs consisted primarily of isolated

© 2005 by Chapman & Hall/CRC

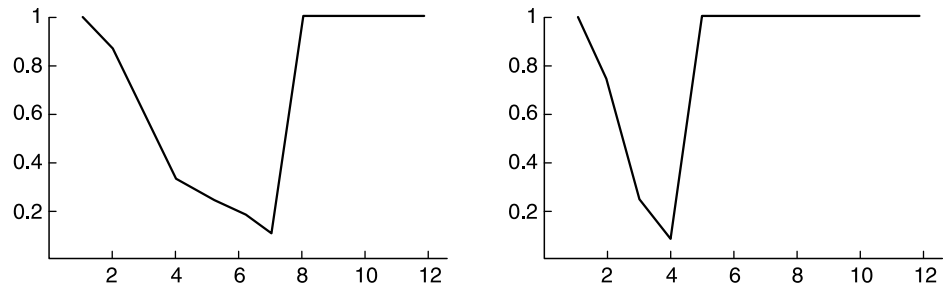


Figure 49.26. Mu[h] (y axis) versus h for Erdös–Rényi graphs of 100 nodes (right) and 130 nodes (left) with p of 10%.

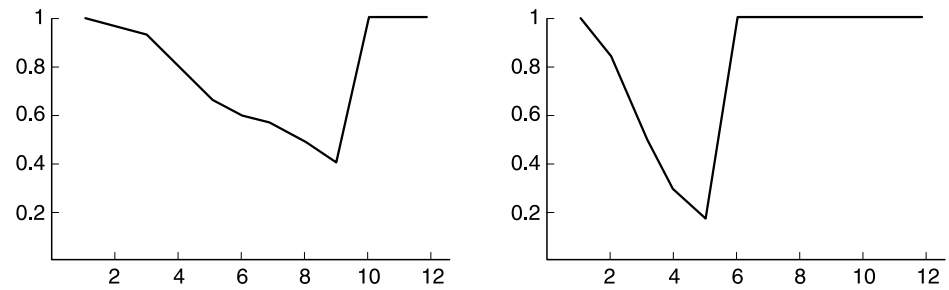


Figure 49.27. Mu[h] (y axis) versus h for Erdös–Rényi graphs of 50 nodes with p of 4% (top) and 8% (bottom).

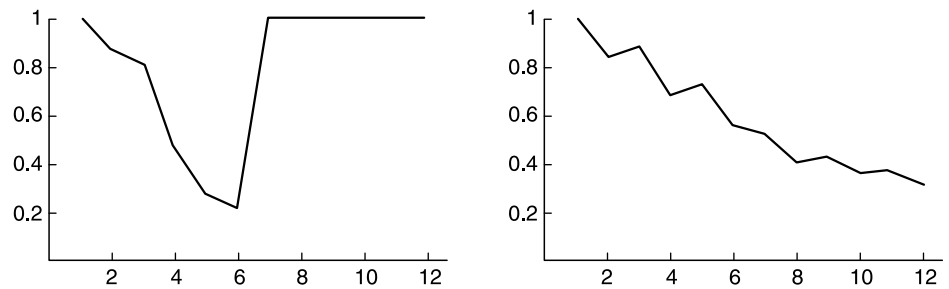


Figure 49.28. Mu[h] (y axis) versus h for small-world graphs of 35 nodes (left) and 135 nodes (right) with cluster size of five and ten edges rewired.

components. This was to be expected, since the expected value of node degree would be less than two. Note how the shape of Mu[h] tends towards a spike at two hops as the edge probability increases.

For small-world graphs there are three parameters to consider: size, cluster size, and number of rewired edges. In contrast to Erdös–Rényi graphs, increasing the graph size causes the shape of Mu[h] to diverge. To illustrate this, look at the curves in Figure 49.28. Similar results were found when the cluster size and number of rewired edges were varied. Increasing the number of edges rewired has the opposite effect, as is shown by Figure 49.29. Similar results were obtained for larger graph and cluster sizes.

Increasing graph cluster size shifts the Mu[h] values to the left, which eventually makes the spike narrower. This is illustrated by Figure 49.30. Similar results occur with other graph sizes and numbers of rewired edges.

© 2005 by Chapman & Hall/CRC

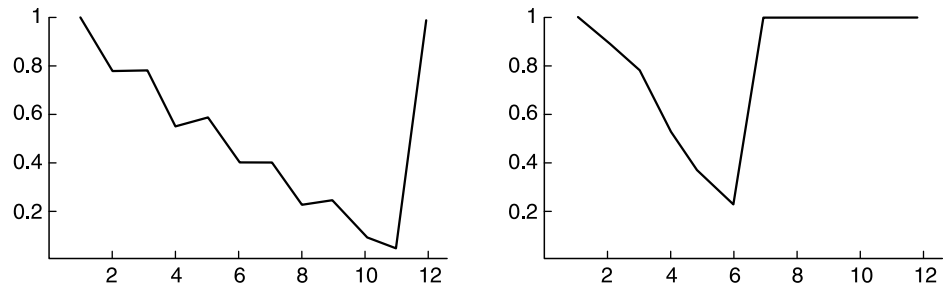


Figure 49.29. Mu[h] (y axis) versus h for small-world graphs of 50 nodes with cluster size of five when four edges (left) and ten edges (right) were rewired.

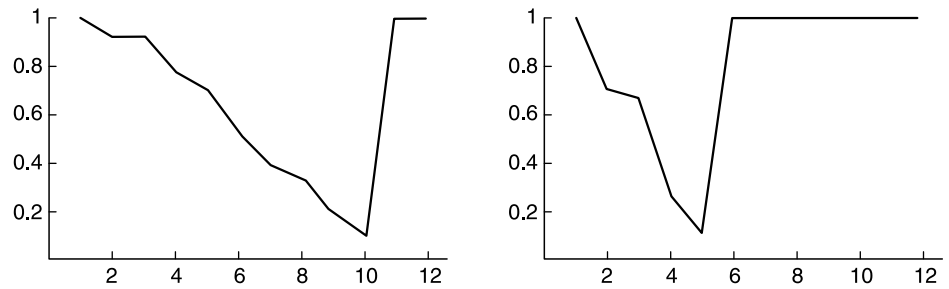


Figure 49.30. Mu[h] (y axis) versus h for small-world graphs of 50 nodes with cluster sizes of four (left) and ten (right) with ten edges rewired.

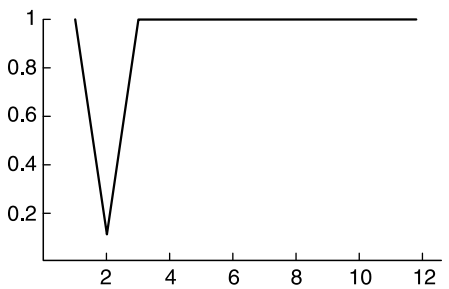


Figure 49.31. Mu[h] (y axis) versus h for scale-free graphs of 50 nodes with average degree of 15.

Scale-free graphs created using the Barabási algorithm have only two parameters: number of nodes and average node degree. The average node degree appears to cause Mu[h] to become narrower as it increases(Figure 49.31). The final result for large node degree looks similar to the Erdös–Rényi results. On the other hand, increasing graph size tends to have no major influence on Mu[h]. Mu[h] for scale-free graphs seems to be narrower and steeper than roughly equivalent small-world and Erdös–Rényi graphs.

49.11 Index Structure

Consider Napster and Gnutella, two extremes of a range of possible P2P designs. How many indexes are desirable for these types of dynamic P2P network? The application domain is a distributed servent sensor network using mobile code to reconfigure itself dynamically. This network is a prototype highly

© 2005 by Chapman & Hall/CRC

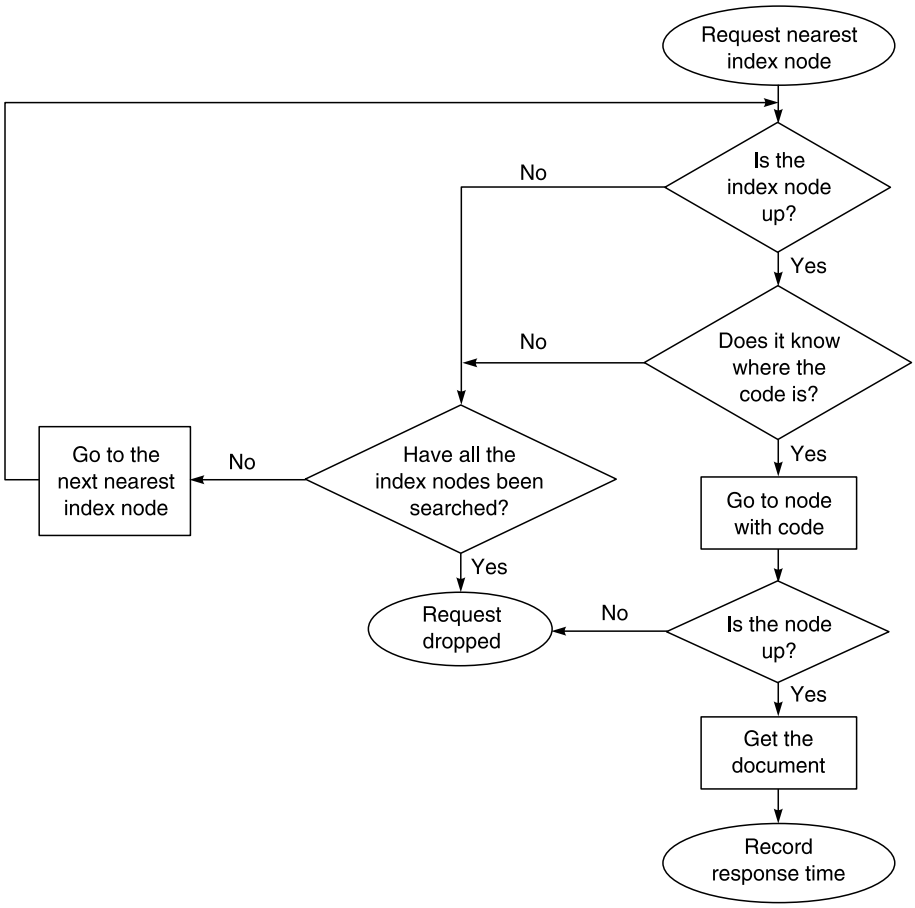


Figure 49.32. Flowchart for finding mobile-code packages in a P2P index.

survivable distributed network service. Figure 49.32 gives a flowchart of the mobile-code indexing system we implemented [20].

Formally: given a network of n nodes, how does the number of indexes affect global system QoS and dependability. QoS issues have been considered by Kapur *et al.* [9]. Here, we analyze network dependability. Specifically, we define dependability to be the probability that an arbitrary request can be completed. Indexes serve subgraphs of approximately equal size created from the original graph. Determining where to place indexes is equivalent to performing these two steps:

1. Perform the k-way partition problem in graph theory, where a graph is partitioned into k (in this case i) partitions of equal size with a minimal number of connections between the partitions. The problem is NP-complete. Many heuristics have been developed to solve this problem for VLSI layout. Known approaches include self-organizing maps and the use of eigenvalues of the connectivity matrix.
2. Place the index at the centroid of each partition.

49.12 Graph Partitioning

We create subgraphs of the P2P network, so that each subgraph is served by its own index. The following conditions for good partitions of the graph are straightforward. The partitions should be of

© 2005 by Chapman & Hall/CRC

equal (or near-equal) size. The majority of the node connections should be within the region served by an index, to support efficient communication between each node and its local index, as well as the subsequent exchanges of mobile-code packages between nodes served by the index. This means that the number of edges between partitions should be minimized. This problem is known as the k-way partition problem, which has been shown to be NP-complete [21]. It is an important problem for many practical applications, such as VLSI design. Many heuristic methods have been formulated for finding approximate solutions to the problem, including self-organizing maps [21] and eigenvalue decompositons of the graph connectivity matrix [22]. We do not consider how the k-way partition of the graph structure is performed. Any established heuristic could be used to derive an approximate answer. We now consider relevant upper and lower bounds for the graph structures produced by the k-way partitioning of the original graph. These bounds describe the resulting subgraphs as structures similar to the original random graphs. This step is essential in determining the number of indexes (partitions) appropriate for the P2P infrastructure and analyzing the effect of partitioning on system performance.

For Erdös–Rényi random graphs, the index partition of the global graph forms a random graph of n/i nodes (discounting rounding errors). Two limiting cases exist for the nodes in the subgraphs:

- Lower bound — same edge distribution probability as the global graph.
- Upper bound — same node degree distribution as the global graph.

Proof. For the lower bound, it suffices to assign nodes to indexes at random. Edges are retained that connect nodes assigned to the same index. Other edges are deleted. Since a random graph is constructed by assigning edges between any two nodes with the same probability, removing any node constructs a random graph with the same probability but with $n-1$ nodes. Creating the k-way partitions is equivalent to performing this process i times, each time removing n/i nodes. Each partition is a random graph with the same edge probability as the global graph, but consisting of only n/i nodes.

For the upper bound, we assume the successful output of the k-way graph partition algorithm. This means that we have k (in this case i) partitions of equal size with a minimal number of connections between partitions. An absolutely optimal partition occurs when there are no connections between partitions. In such a case, no edges would be removed in creating the partitions. The node degrees are unchanged and the degree distribution for the global graph is unchanged. The assignment of nodes to subgraphs is unbiased. The nodes chosen should have the same distribution as the global graph. The change in sample size would significantly modify the variance. Because the global graph is connected, this case cannot occur in practice.

Since the number of connections between partitions is minimal, we expect the actual partitioning behavior to be closer to the upper bound as long as the number of indexes is small relative to the number of nodes. As the number of indexes approaches n, the lower bound is likely to be a better approximation.

For scale-free graphs, the upper bound for the index partitions is the same as for the random graph and the same proof is valid. Since the probability of an edge existing between two nodes is not defined for scale-free graphs, the lower bound suggested for random graphs is undefined and inappropriate. Instead, we use a lower bound defined by assigning nodes to indexes at random and assume that all edges are equally likely to cross index boundaries. This lower bound is a scale-free network, but the node degree in the subnetwork will, on average, be a fraction $(1/i)$ of the degree in the global network. This is a lower bound in part because it ignores clustering effects.

Proof of lower bound. For the lower bound, consider a total of n nodes in the network. n/i nodes are in the subgraph served by an index. If an edge leaving a node is equally likely to lead to a node served by any given index, then the likelihood of it reaching any given index is $1/i$. The likelihood of the subgraph not being the one the current node is in is $1 - 1/i$ or $(i-1)/i$. A node with degree d would, therefore, have an expected degree of d/i.

For *small-world graphs*, two distinct cases exist for the structure inside the index partitions: (i) fewer nodes in the subgraph than in the initial fully connected components; (ii) more nodes in the subgraph

© 2005 by Chapman & Hall/CRC

than in the fully connected components. For both cases, the subgraphs will be constructed by removing some of the small fraction of edges that have been rewired. The subgraph will, therefore, keep the global structure of the small-world graph. It will be a graph that consists of a set of almost fully connected components connected by a small number of random vertices. Note that, depending on the value of i, if $n \bmod i \neq 0$, then some fully connected components will be split between indexes. This is not a significant modification of the structure, since within the split component almost all nodes will almost always be within a single hop of each other. In addition, all nodes within a fully connected component are within one hop of each other. If the graph is made from fully connected components of size c, then the subgraph would consist of approximately n/ic fully connected components connected at random. How the random connections are handled depends on the number of edges rewired at random.

49.13 Expected Number of Hops

For P2P infrastructure analysis, like the QoS study of Kapur *et al.* [9], an important issue is the expected number of hops between nodes in the network. In this section we present two analyses of this problem. One analysis is based on an empirical study. The other uses the connectivity matrices to compute the expected number of hops directly. This analysis is important for designing both wired and wireless sensor network systems.

To measure the rate of growth of the expected number of hops as a function of graph size, Dorogovtsev and Mendez [23] derive the following simple estimator for the average number of hops between any two nodes in an Erdös–Rényi random graph:

$$q_{\text{ave}} \approx \frac{\ln[n]}{\ln[q_1]}$$

showing that the average number of hops grows as the logarithm of graph size. Empirical results from Watts [2] indicate that the relationship between graph size and average path length is similar for random and small-world graphs. Scale-free graphs appear to grow even more slowly [13,19]. Our empirical tests support this assumption.

Newman *et al.* [24] provide a derivation using generating functions that shows that the number of nodes h hops away can be approximated using only two factors: the number of nodes one hop away and the number of nodes two hops away. The results are approximate since detailed graph structures cause graph instances to deviate, and they assume a single fully connected component in the graph. According to them (using our notation):

$$q_h = \left(\frac{q_2}{q_1}\right)^{h-1} q_1 \tag{49.20}$$

The generating function approach ignores the structure of an individual graph instance. Newman [17] provides a more accurate approach. A different derivation follows that produces a similar result. The result here is proposed for a slightly different problem and is slightly different. We use a recursion equation to find a simple estimate of the number of nodes reachable after exactly h hops.

The average number of nodes reachable after one hop is by definition d_{ave} (i.e. q_1). For the second hop, every node reached has, on average, $d_{\text{ave}} - 1$ degrees free. By definition C percent of those $d_{\text{ave}}(d_{\text{ave}} - 1)$ degrees were already reached in one hop, leaving $(1 - C)$ percent connecting to new nodes. Mu[2] compensates for the presence of quadrilaterals in the graph. Mu[h] generalizes this. We apply the same logic to define q_3 and so on. The resulting equation is

$$\begin{aligned} q_h &= (d_{\text{ave}} - 1) q_{h-1} (1 - C) \text{Mu}[h] \\ q &= d_{\text{ave}} \\ \text{for } h > 1 \quad q_h &= (1 - C)^{h-1} (d_{\text{ave}} - 1)^{h-1} d_{\text{ave}} \prod_{i=2}^{h} \text{Mu}[i] \end{aligned} \tag{49.21}$$

© 2005 by Chapman & Hall/CRC

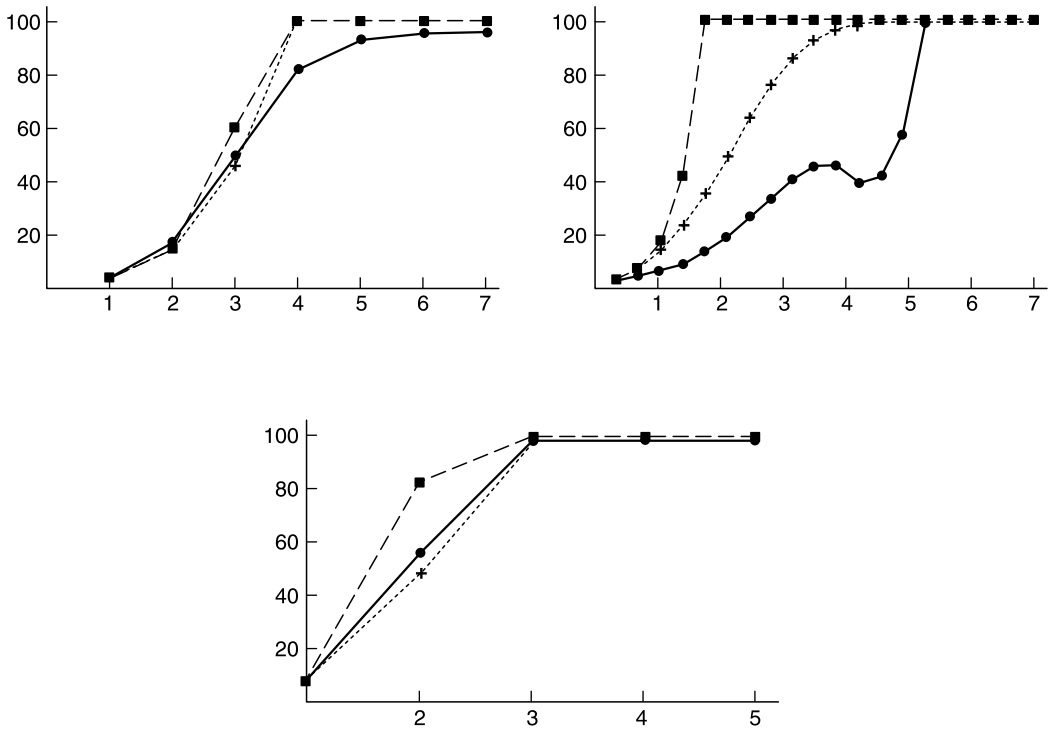


Figure 49.33. Average number of nodes reachable in h hops (y axis) versus h hops (x axis) for Erdös–Rényi (top left), small-world (top right), and scale-free graphs (bottom). Solid line is results from 35 appropriately generated random graphs with 95% confidence interval. Dashed line is from the Newman [17] estimator. Dotted line is the estimate from Equation (49.22).

A more accurate estimate of the number of nodes reachable in h hops is achieved by taking into account the greater likelihood of connecting to a node with a higher degree [17] (again, $h > 1$; for $h = 1$ Equation (49.21) holds):

$$q_h = (1 - C)^{h-1} d_{\text{ave}} \prod_{i=2}^{h} \text{Mu}[i] \left(\sum_{d=1}^{n-1} d(d-1)^{h-1} p_d \right) \tag{49.22}$$

Figure 49.33 shows the relative quality of these estimators, using the same sets of graphs used to estimate the Mu[h] factors. The solid lines have error bars and show the mean number of nodes reached after the given number of hops. The line connected by alternating dashes and dots is the generating function estimate. The dotted line shows the estimate function in Equation (49.22). To calculate the estimates for the class of graphs, we computed the C and Mu[h] factors for each graph and used the mean value in the estimation functions. Note that all three graphs have the same general shape. We also plotted the values for individual graphs and the results were similar. The actual plot is an s-curve that approaches 100 fairly rapidly. The generating function estimate consistently overestimates the number of nodes reachable within h hops.

The top left graph of Figure 49.33 is the Erdös–Rényi data. Note that the actual plot does not reach 100 nodes, because some of the graphs generated were not fully connected. There was always one major component, but some nodes were not connected to it. The top right graph is for the small-world graphs. Interestingly, underestimation is most pronounced in this case. The bottom graph shows results for the

© 2005 by Chapman & Hall/CRC

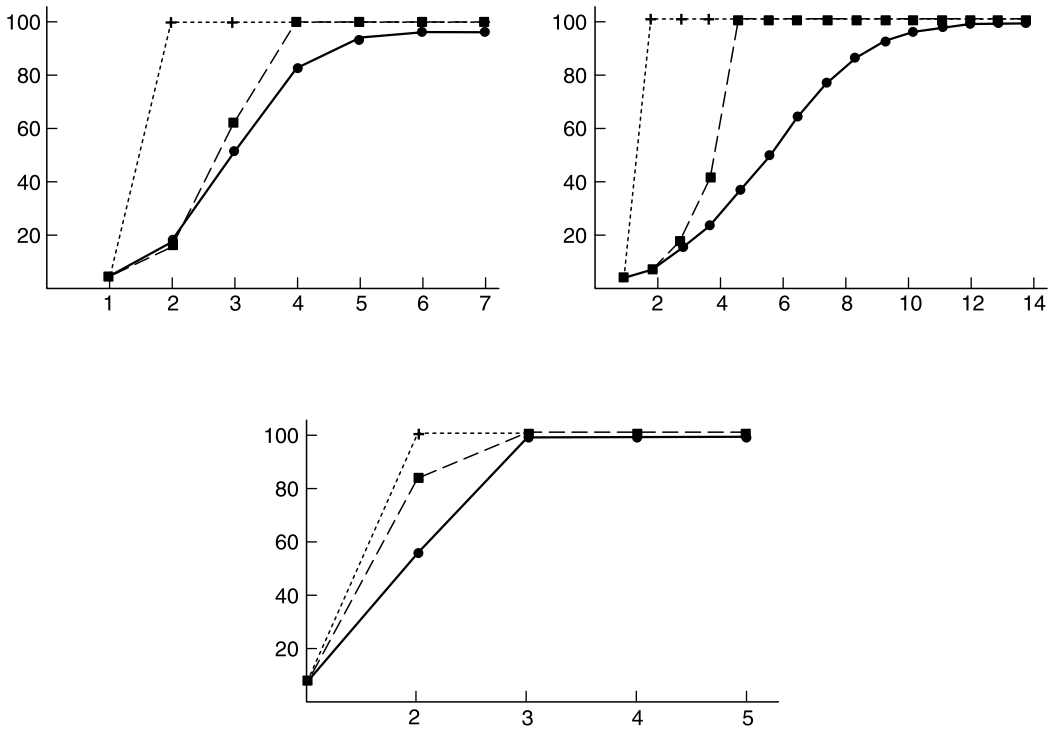


Figure 49.34. Average number of nodes reachable in h hops (y axis) versus h hops (x axis) for Erdös–Rényi (top left), small-world (top right), and scale-free graphs (bottom). Dashed line is from the Newman [17] estimator with Mu[1]=1. Dotted line estimate is Equation (49.22). For the two estimators, all Mu[h] values were set to one.

scale-free graphs. The estimate using C and Mu[h] initially underestimates the number of nodes. It converges to the generating function estimate for Erdös–Rényi and scale-free graphs. For the small-world graphs the estimating function derived in this chapter has an entirely different shape.

To decouple the effects of the C and Mu[h] factors, Figure 49.34 shows the same plots with all Mu[h] values set to unity. Note that underestimation no longer occurs. The estimate is worse than that provided by the generating function.

The expected number of hops between any two nodes chosen at random can now be calculated as

$$z_{\exp} = \frac{q_1 + 2(q_2 - q_1) + \cdots}{n}$$

$$z_{\exp} = \left(\frac{1}{n}\right)\left(\begin{array}{l} d_{\text{ave}} + 2\left[(1-C)\text{Mu}[2]\left(\sum_{d=1}^{n-1} d(d-1)p_d\right)\right] \\ + \sum_{h=3}^{\text{max hops}} h\left\{\left[(1-C)^{h-1}\prod_{i=2}^{h}\text{Mu}[h]\left(\sum_{d=1}^{n-1} d(d-1)^{h-1}p_d\right)\right]\right. \\ \left. -\left[(1-C)^{h21}\prod_{i=2}^{h-1}\text{Mu}[h]\left(\sum_{d=1}^{n-1} d(d-1)^{h-2}p_d\right)\right]\right\} \end{array}\right) \tag{49.23}$$

Note that we estimate the number of hops reachable in exactly h hops z_h as $q_h - q_{h-1}$, the set of nodes reachable in h hops minus the set of nodes reachable in $h-1$ hops. Since not all nodes reachable in $h-1$ hops are reachable in h hops, this is not an exact measure. As we will see, it functions fairly well in practice, although it causes a slight undercount.

© 2005 by Chapman & Hall/CRC

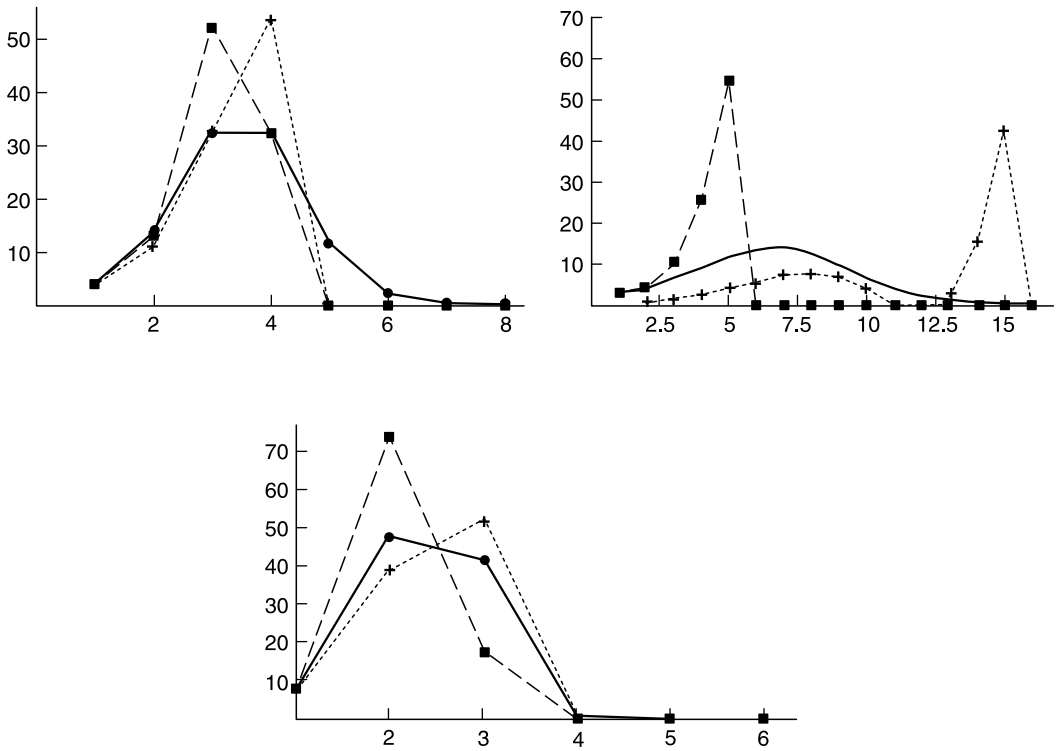


Figure 49.35. Number of nodes first reachable in h hops (y axis) versus h hops (x axis) for Erdös–Rényi (top left), small-world (top right), and scale-free graphs (bottom). Dashed line is from the Newman [17] estimator. Dotted line is the estimate from Equation (49.23).

As before, we plot the results from our graph test set versus the two estimators. The results are consistent with the cumulative graphs. As should be expected, the discrepancies are more notable. For Erdös–Rényi graphs the estimation is reasonable. For scale-free the estimate is very good. For small-world graphs the results are disappointing. The estimate is consistently low. A spike of nodes at the end compensates for this.

To verify the ability of these estimators to predict the expected number of hops in a random graph, we compare the actual expected number of hops between two nodes selected at random in our set of 35 test cases (Figure 49.35). The expected values are: 3.31463 (Erdös–Rényi), 4.39716 (small world) and 2.36894 (scale free). The generating function predictions are 3.24785 (Erdös–Rényi), 3.59476 (small world) and 2.09639 (scale free). The predictions from the derived estimation function are 3.52925 (Erdös–Rényi), 6.46731 (small world) and 2.47913 (scale free). Both estimators function well. The generating function underestimates the expected number of hops. Equation 49.23 overestimates the number of hops. The only significant deviation appears to occur using the estimator function that includes clustering and mutuality parameters for small-world graphs.

We can also use the probabilistic connectivity matrices to estimate the expected number of hops between nodes. To do so, we use the following theorem.

Theorem 49.1. *Element (j, k) of M^z is the probability that a walk of length z exists between nodes j and k.*

Proof. The proof is by induction. By definition, each element (j, k) is the probability of an edge existing between nodes j and k. M^2 is the result of multiplying matrix M with itself. Equation (49.16) is used to calculate each element (j, k), since all values are probabilities. As explained in Section 49.7, this

© 2005 by Chapman & Hall/CRC

calculates the probability of a path of length two existing between nodes j and k by exhaustively enumerating the likelihood of the path passing through each intermediate node in the graph. Using the same logic, M^z can be calculated from M^{z-1}using matrix multiplication to consider all possible intermediate nodes between nodes j and l, where M^{z-1} has the probabilities of a walk of length $z-1$ between j and k, and M has the values defined previously.

Example. Probabilities of walks of length three in an Erdös–Rényi graph of four nodes for $p = 0.6$ and 0.65:

$$M = \begin{bmatrix} 0 & 0.65 & 0.65 & 0.65 \\ 0.65 & 0 & 0.65 & 0.65 \\ 0.65 & 0.65 & 0 & 0.65 \\ 0.65 & 0.65 & 0.65 & 0 \end{bmatrix} \quad M^2 = \begin{bmatrix} 0 & 0.666 & 0.666 & 0.666 \\ 0.666 & 0 & 0.666 & 0.666 \\ 0.666 & 0.666 & 0 & 0.666 \\ 0.666 & 0.666 & 0.666 & 0 \end{bmatrix}$$

$$M^3 = \begin{bmatrix} 0 & 0.679 & 0.679 & 0.679 \\ 0.679 & 0 & 0.679 & 0.679 \\ 0.679 & 0.679 & 0 & 0.679 \\ 0.679 & 0.679 & 0.679 & 0 \end{bmatrix} \tag{49.24}$$

$$M = \begin{bmatrix} 0 & 0.6 & 0.6 & 0.6 \\ 0.6 & 0 & 0.6 & 0.6 \\ 0.6 & 0.6 & 0 & 0.6 \\ 0.6 & 0.6 & 0.6 & 0 \end{bmatrix} \quad M^2 = \begin{bmatrix} 0 & 0.59 & 0.59 & 0.59 \\ 0.59 & 0 & 0.59 & 0.59 \\ 0.59 & 0.59 & 0 & 0.59 \\ 0.59 & 0.59 & 0.59 & 0 \end{bmatrix}$$

$$M^3 = \begin{bmatrix} 0 & 0.583 & 0.583 & 0.583 \\ 0.583 & 0 & 0.583 & 0.583 \\ 0.583 & 0.583 & 0 & 0.583 \\ 0.583 & 0.583 & 0.583 & 0 \end{bmatrix} \tag{49.25}$$

Computing the expected number of nodes reachable in r hops is very straightforward in this case. If the graph description is regular (all rows are permutations of the same values), then predicting this value is simple. First, compute M^r. The sum of the values of any row provides the expected number of nodes reachable in r hops. This value is the same for every node.

For scale-free graphs and small-world graphs whose descriptions are not regular, compute the sum for each row. The maximum and minimum values of the sum provide the range of possible values. The average can easily be computed, and is a reasonable solution for small-world graphs. Since the degree of hub nodes in scale-free graphs is much larger than for most nodes (when γ is significantly greater than one), the mode value of the row sum is likely to be a better estimate for scale-free graphs.

49.14 Probabilistic Matrix Characteristics

The applications we have found for the probabilistic connectivity matrix representation are mainly derived from the fact that each element l_{ij} of a connectivity matrix L raised to the power k expresses the number of walks from node i to node j. Since our matrix is probabilistic, instead of computing the number of paths using matrix multiplication we compute the probability that a path exists using a similar approach. A walk from i to j of two hops exists when there is an intermediate node m with edges from i to m and m to j. In this case, the probability of this path existing is the product of elements $l_{l,m}$

© 2005 by Chapman & Hall/CRC

and $l_{m,j}$ of the probabilistic connection matrix. The total probability of the path existing is computed by calculating the inclusive-or for all intermediate nodes. A convenient way of computing this is computing the complement of the probability that all paths do not exist:

$$l_{ij}^{h} = 1 - \prod_{k=1}^{n} \left(1 - l_{ik}^{h-1} l_{kj}\right) \tag{49.26}$$

This can be implemented by taking a standard matrix multiplication implementation and replacing the summation of products in the inner loop with Equation (49.26). Note that we constrain diagonal values to remain zero.

We illustrated how to construct these matrices for important graph classes. We now discuss the structure and meaning of the matrices. By definition, connectivity matrices are square with the numbers of rows and columns both equal to the number of vertices in the graph (n). Each element (j, k) is the probability of an edge existing between nodes j and k. Since we consider only non-directed graphs, (j, k) should equal (k, j). Care should be taken to guarantee that algorithms for constructing matrices provide symmetric results.

Theorem 49.2. *The sum of each row (column) of the probabilistic connectivity matrix M provides the expected degree of every node in G.*

Proof. The expected value of a random variable is defined as the sum of possible values of the random variable times the probability of the value. Each element (j, k) in row j is an independent Bernoulli random variable expressing the likelihood of an edge existing between nodes j and k. The expected number of edges between j and k is the value of (j, k) times one. The expected degree of a node is the expected number of edges with the node as an end point. The expected degree of node j is the sum of the number of edges connecting j with all other nodes. Since each element (j, k) of row j expresses the likelihood of an edge between j and k (k ranging from 1 to n), the expected degree of j is the expected value of the sum of k Bernoulli random variables with probabilities of value (j, k). Since the expected value of a sum of random variables is the sum of the expected values of the random variables, the theorem must be true. QED.

Theorem 49.3. *M is a probabilistic connectivity matrix of random graph class G. We define rpm-graphs as graph classes where each node has equivalent probabilities of being connected to other nodes. If a graph class defines rpm-graphs, then all nodes in G are equivalent and every row (column) is a permutation of every other row (column).*

Proof. A regular graph is one in which all nodes have the same degree. Cvetkovic *et al.* [11] provide a proof that each row of the connectivity matrix of a regular graph is a permutation of every other row. This proof also applies to our definition of rpm-graphs, with the exception that elements of the connectivity matrices for rpm-graphs do not have the value one or zero, but are in the range from one to zero. In an rpm-graph class, because each row (column) is a permutation of every other row (column), the expected value of the degree of all nodes is equal. Note that instances of the rpm-graph class are almost certainly not regular graphs.

Theorem 49.4. *For an rpm-graph class G, the value of the eigenvalue with the largest magnitude of its associated connectivity matrix is the expected degree of each node in G.*

Proof. This proof follows directly from the proof of this assertion for nonrandom graphs given by Cvetkovic *et al.* [11].

© 2005 by Chapman & Hall/CRC

49.15 Network Redundancy and Dependability

Given the expected number of hops in terms of the number of nodes and degree distribution, we consider the question of approximating dependability for paths of h hops for random graphs with arbitrary degree distributions using the same statistics. We start by assuming each link has the same probability of failure f. Note that, ignoring node failures, the dependability a of a serial path through the graph can be calculated using the fact that it can only be working if all components work. This is given by (see Figure 49.36):

$$a_h = (1 - f)^h \tag{49.27}$$

where a_h is the dependability of an h hop serial system. This can be compared with the dependability of a parallel system of h components. In which case, the system fails only if all h components fail concurrently. Using the same variables a and f as previously, this gives (see Figure 49.37).

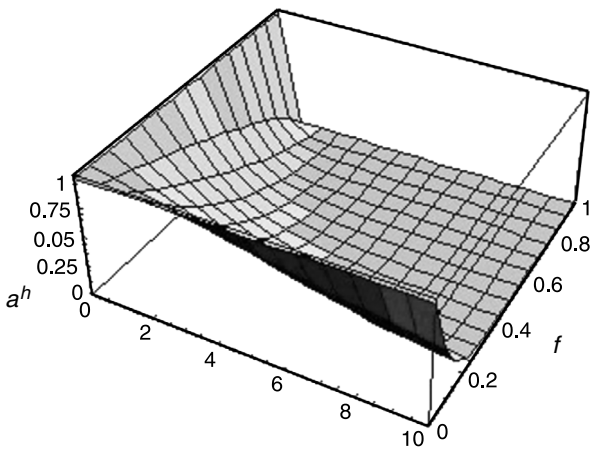


Figure 49.36. The availability a^h (vertical axis) of an h-hop serial system. The number of hops h varies from zero to ten (front axis) and the probability of failure f varies from zero to one (side axis).

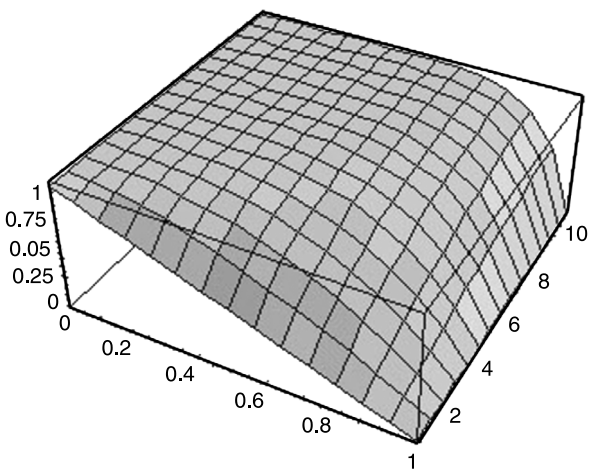


Figure 49.37. System dependability (z axis) for a parallel system versus number of components (y axis) and component dependability (x axis). This is a graphical representation of Equation (49.27).

© 2005 by Chapman & Hall/CRC

Figures 49.36 and 49.37 illustrate well-established facts. For serial systems, as the number of components increase, so the system dependability decreases rapidly and asymptotically approaches zero. The redundancy in parallel systems causes dependability to increase quickly and asymptotically approach one as the number of components increases. Obviously, degradations in component dependability adversely affect both approaches.

All classes of random graph structures we consider can have nonnegligible amounts of redundancy. We now analyze how this redundancy affects communications between nodes in the graph structures. Ideally, the statistics we have discussed should enable us to predict path dependability in these systems.

We have established ways of dividing random and pseudo-random graphs into partitions for indexing, authentication, etc. When this is done, we have parameters that describe the graph. Given these parameters, we can calculate statistics that describe clustering and mutuality in the graph. We are also able to estimate the expected number of hops required to service a request for a mobile-code package [9]. We now attempt to use this information to determine how partitioning the network affects dependability.

To determine system dependability as a function of the number of indexes (partitions), we need only calculate communications dependability as a function of the number of hops. We modify a to determine a new factor $\hat{a}$ which takes into account the network redundancy. Network redundancy provides multiple possible paths between two neighboring nodes. The single-hop dependability adjusted to consider alternate paths of exactly r hops is $\hat{a}_r$, giving the following:

$$\hat{a} = 1 - \prod_{r=1}^{n-(h+1)} (1 - \hat{a}_r) \tag{49.29}$$

Thus, a better estimate of the dependability of an h-hop path would be given by $\hat{a}^h$. Before proceeding, let us analyze what redundancy means in terms of availability. Figure 49.38 plots the dependability of a one-hop system with no redundancy and the dependability of a 50-node system where independent paths of all lengths are guaranteed to exist versus a probability of single link failure progressing from one to zero. Figure 49.39 shows the difference between the two. Note that the maximum dependability gain (about 25%) occurs at a probability of failure around 40%.

We modify Equation (49.29) by considering redundant paths. Multiple paths of varying lengths between two nodes frequently exist in these graph classes, and they can be used as alternatives to the shortest path of length h connecting the nodes. We attempt to quantify this effect by computing the expected amount of redundancy for individual edges (one hop paths) in the graph. Methods for computing the likelihood of redundant paths of varying lengths are described. These probabilities are

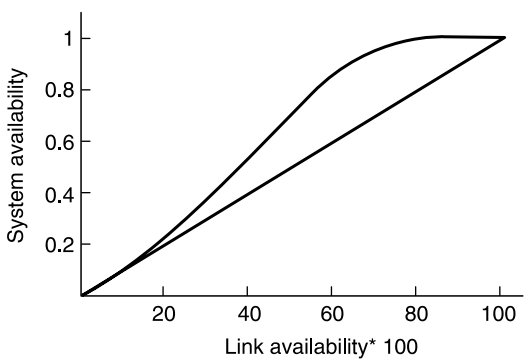


Figure 49.38. System dependability (y axis) versus component dependability times 100 (x axis). The bottom line is for a one-component system. The top line is for a system of 50 components with redundant links between all nodes.

© 2005 by Chapman & Hall/CRC

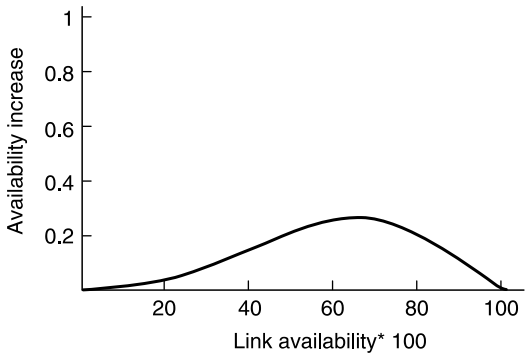


Figure 49.39. System dependability gain (y axis) versus component dependability times 100 (x axis) for a fully redundant system of 50 nodes.

then used to calculate a more realistic availability estimate for the h-hop path using the modified per hop availability.

The variable $\hat{a}$ is used for the per hop availability of an edge, taking into account redundancy in the random graph. From Equation (49.27), a^r is the dependability of a path of r hops with no redundancy. Let us consider two adjacent nodes i and j. Equation (49.29) defines $\hat{a}_r$ the dependability of a path of r hops connecting nodes i and j parallel to the edge (path of unit length) connecting the nodes. Equation (49.30) takes into account the probability that this path exists:

$$\hat{a}_r = a^r P[q_r|q_1] \tag{49.30}$$

where $P[q_r|q_1]$ is the probability that a path of r hops exists between two nodes given that a path of one hop (an edge) exists connecting the nodes. We now have paths of differing lengths connecting two nodes in parallel. Remembering that h is the expected number of hops between two nodes in the network, our estimate of the dependability of the file transfer path becomes $\hat{a}^h$. This estimate has two weak points: (i) it is pessimistic, in that it only considers redundancy for a single hop; (ii) it is optimistic, in that it makes a tacit assumption that the paths of differing lengths share no edges.

For the sake of brevity, we introduce notation $P_{x,\ldots,y}$ for $P\lfloor\{q_x \wedge \cdots \wedge q_y\}\rfloor$. For example, $P\big[\{q_3 \wedge q_2 \wedge q_1\}\big]$ (the probability a node belongs to the set of nodes where paths of lengths 3, 2 and 1 connect it to another node) can be expressed as $P_{3,2,1}$, and $P_{2,\neg 1}$ means $P\big[\{q_2 \wedge \neg q_1\}\big]$ (the probability that a node belongs to the set of nodes with a path of length 2 and none of length 1 connecting it to another node).

Consider the effect of two-hop paths:

$$\hat{a}_2 = a^2 P[q_2|q_1] \tag{49.31}$$

where $P[q_2|q_1]$ is the probability that a path of two hops exists between two nodes given that an edge connects the nodes. This differs subtly from the cluster coefficient. C is the percentage of two hop paths already reached in one hop. $P[q_2|q_1]$ is the percentage of one-hop paths that also yield two-hop paths. It can be computed by modifying the method for computing cluster coefficient given in Section 49.9. It suffices to swap the roles of M^2 and M.

The variable $\hat{a}_3$ depends on $P[q_3|q_1]$, which we decompose into distinct cases depending on the existence (or absence) of paths of length 2:

$$\begin{aligned}\hat{a}_3 &= a^3 P[q_3|q_1] = a^3 \frac{P_{3,1}}{P[q_1]} \\ &= \frac{(1-f)^3}{P[q_1]} \left(P_{3,2,1} P[q_2] + P_{3,\neg 2,1} P[\neg q_2]\right)\end{aligned} \tag{49.32}$$

© 2005 by Chapman & Hall/CRC

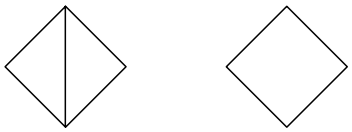


Figure 49.40. Left: figure implied by elements of the set $\{q_3 \wedge q_2 \wedge q_1\}$. Right: figure implied by elements of $\{q_3 \wedge \neg q_2 \wedge q_1\}$.

P$[q_2]$ is q_2/n and $P[q_1]$ is q_1/n. Figure 49.40 shows the figures implied by the factors in Equation (49.32). The right-hand figure is two contiguous triangles and occurs with probability $P[q_2|q_1]^2$.

The left-hand object in Figure 49.40 is a rectangle. The probability of this object occurring can be calculated by the following procedure:

- First calculate M^2.
- Elements of $M^2 > 1$, with corresponding element of $M =$ zero, are rectangle vertices.
- $P_{3,\neg 2,1}$ is the number of columns of M^2 with elements that meet this criterion divided by n.

This procedure provides the final information needed to compute $\hat{a}_3$. Equation (49.32) can be completed, giving

$$\hat{a}_3 = \frac{(1-f)^3}{q_1}\left[P_{3,2,1}q_2 + P_{3,\neg 2,1}(n - q_2)\right] \tag{49.33}$$

All $P_{i,\neg(i-1),\ldots,\neg 2,1}$ where the number of hops i is odd describe the probability of the node being a vertex of a polygon with an even number of sides $2(i-1)$. All sides are edges of the graph. Figure 49.41 illustrates this. The probability can be computed by extending procedure 4 to give procedure 5.

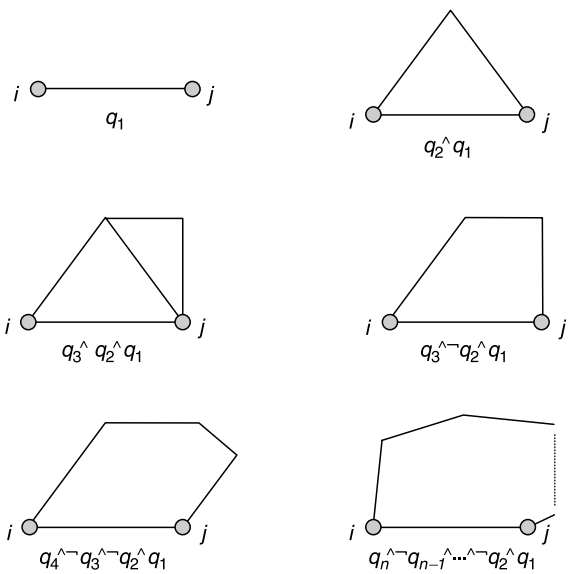


Figure 49.41. Geometric figures implied by the existence of paths of different lengths. If paths of length q_n and q_{n+1} exist, then a triangle is implied. Paths of length q_i and q_j with $i > j$ and no paths of length k, $i > k > j$, imply a polygon with $(i-j)+2$ sides.

© 2005 by Chapman & Hall/CRC

The extensions in step 2 of procedure 5 are needed to counteract the presence of loops in the calculation of powers of the connectivity matrix M.

Estimation of $P_{i,\neg(i-1),\ldots,\neg 2,1}$:

1. Initialize L' to the connectivity matrix L.
2. Perform the following for $i-1$ iterations:
   1. Initialize L'' to equal L'
   2. Set diagonal elements of L' to zero
   3. Set all nonzero elements of L' to one
   4. Set all elements $l_{i,j}$ of L' to $l_{i,j} * l_{j,i}$
   5. Multiply L' by L giving L'
3. Count the columns of L' with elements greater than one and the corresponding elements of L^{i-1} to L equal to zero.
4. Divide this sum by n giving $P_{i,\neg(i-1),\ldots,\neg 2,1}$.

These values can be computed using an instance of the random graph class. As with $M[h]$, a more reliable value is found by computing the mean over several instances of the graph classes. If the number of hops is even, then the procedure 5 will not work. In which case, estimate $P_{i,\neg(i-1),\ldots,\neg 2,1}$ by averaging $P_{i+1,\neg i,\ldots,\neg 2,1}$ and $P_{i-1,\neg(i-2),\ldots,\neg 2,1}$.

To compute $\hat{a}_n$ we follow a similar approach to computing $\hat{a}_3$. The value of $\hat{a}_n$ is $(1-f)^n\, P[q_n|q_1]$, and:

$$
\begin{aligned}
P[q_n|q_1] &= \frac{P_{n,n-1,1}P[q_{n-1}] + P_{n,\neg n-1,1}P[\neg q_{n-1}]}{P[q_1]} \\
P_{n,n-1,1} &= P_{n,n-1,n-2,1}P[q_{n-2}] + P_{n,n-1,\neg n-2,1}P[\neg q_{n-2}] \\
P_{n,\neg n-1,1} &= P_{n,\neg n-1,n-2,1}P[q_{n-2}] + P_{n,\neg n-1,\neg n-2,1}P[\neg q_{n-2}]
\end{aligned}
\tag{49.34}
$$

Equation (49.34) is recursive, and continues until each element is a sequence exhaustively enumerating the existence or nonexistence of all paths of length 1 to r. The probability of each atomic element is the product of the probabilities of the polygons whose union defines the object. For example, $P_{4,\neg 3,2,1}$ describes the union of a triangle and a rectangle. It is equal to $P[q_2|q_1]$ times $P_{3,2,1}$.

When the recursion of Equation (49.34) terminates, each factor contains a variable of the form $P_{j,\ldots,1}$, where j and 1 delimit a list of path lengths, some of which are negated. The limits of the list $(j, 1)$ are never negated. This term needs to be translated into a probability.

Calculation of $P_{j,\ldots,1}$:

1. Initialize probability $P_{j,\ldots,1}$ to 1.
2. Start with 1 and count the number of negated paths until the next nonnegated one. Call this number k.
3. If $k=0$, then the polygon described is a triangle. $P_{j,\ldots,1}$ becomes $P_{j,\ldots,1}\, P[q_2|q_1]$.
4. If $k>0$, then the polygon described has $3+k$ sides. The probability of it being a part of the figure described by $P_{j,\ldots,1}$ is described by $P_{k/2,\neg(k/2-1),\ldots,\neg 2,1}$. We have already described how to estimate this value for even and odd k. $P_{j,\ldots,1}$ becomes $P_{j,\ldots,1}\, P_{k/2,\neg(k/2-1),\ldots,\neg 2,1}$.
5. Replace one with the next nonnegated path length. If that path length is less than j, then start again at step 2. Else, terminate the calculation.

We now compute an estimate for $\hat{a}$, the expected value of the dependability of an edge in a random graph. Considering the system as a set of parallel paths of differing lengths gives

$$
\hat{a} = 1 - \prod_{j=1}^{\text{diameter}} \left(1 - \hat{a}_j\right) \tag{49.35}
$$

The path dependability for the h-hop path becomes $\hat{a}^h$.

© 2005 by Chapman & Hall/CRC

This method has several shortcomings:

- It implicitly assumes independence for paths through the graph.
- Computation of Equation (49.34) has a combinatorial explosion. For paths of length j, 2^j factors need to be considered.
- Tables of $P_{i,\ldots,1}$ statistics are not readily available. The estimates we describe are computable, but computation requires many matrix multiplications. Stable statistics require computation using several graph instances. For large graphs, the amount of computation required is nonnegligible.
- It ignores the existence of multiple redundant paths of the same length increasing per hop dependability. As shown in Figure 49.39, this factor is important.

Most of these can be overcome by realizing that the additional dependability afforded by a path drops off exponentially with the number of hops. It should be possible to stop the computation when a^r becomes negligible. Another factor to consider is that the diameter of the graph scales at worst logarithmically for these graph classes. The algorithm scales as the exponential of a logarithm, making it linear. This approach reveals the redundancy inherent in these systems and how it can be effectively exploited.

Alternatively, $P[q_r]$ can be taken directly from the probabilistic connectivity matrices. For the Erdös–Rényi graph all nondiagonal elements have the same value, which is $P[q_h]$. For small-world and scale-free graphs, the average of all elements in M^h is a reasonable estimate of q_h. We suggest, however, using the minimum nondiagonal element value in the matrix. The average will be skewed by large values for connections in the clusters for small-world graphs and large values for hub nodes in scale-free graphs. The minimum value is the most common in both graph classes and provides a more typical estimate of the probability of connection between two nodes chosen at random.

49.16 Vulnerability to Attack

Empirical evidence that the Internet is a scale-free network with a scaling factor close to 2.5 is discussed by Albert and Barabási [13]. Albert *et al.* [3] analyze the resiliency of the Internet to random failures and intentional attacks using a scale-free model. Simulations show that the Internet would remain connected even if over 90% of the nodes fail at random, but that the network would no longer be connected if only 15% of the best-connected hub nodes should fail. In this section we show how this problem can be approached analytically. The techniques given here allow an analytical approach to the same problem:

- Construct a matrix that describes the network under consideration.
- The effect of losing a given percentage of hub nodes can be estimated by setting all elements in the bottom j rows and left j columns to zero, where j/n approximates the desired percentage.
- Compute C^2 and see whether the probabilities increase or decrease. If they decrease, the network will partition.
- Find the percentage where the network partitions.

Theorem 49.5. *The critical point for scale-free network connectivity arises when the number of hub nodes failing is sufficient for every element of the square of the connectivity matrix to be less than or equal to the corresponding element in the connectivity matrix.*

Proof. Hub nodes correspond to the last rows and columns in the probabilistic connectivity matrix. When a hub node fails, all of its associated edges are removed from the graph. This is modeled by setting all values in the node's corresponding row and column to zero. Matrix multiplication is monotone decreasing. If all elements of matrix K are less than all elements in matrix K', then, for any matrix J, $JK < JK'$. When all two-hop connections are less likely than one-hop connections, then three-hop connections are less likely than two-hop connections, etc. Using the same

© 2005 by Chapman & Hall/CRC

logic as with Erdös–Rényi graphs, this will cause the network to tend to be disconnected. Therefore, when enough hub nodes fail so that all elements of M^2 are less than the corresponding elements in M the corresponding networks will be more likely to be disconnected. QED.

49.17 Critical Values

Many graph properties follow a 0–1 law. The property either appears with probability zero or probability one in a random graph class, depending on the parameters that define the class. Frequently, an abrupt phase transition exists between these two phases [6,12]. The parameter value where the phase transition occurs is referred to as the critical point. The connectivity matrices defined in this chapter can be useful for identifying critical points and phase transitions.

Theorem 49.6. *For Erdös–Rényi graphs of n nodes and probability p of an edge existing between any two nodes, the critical point for the property of graph connectivity occurs when* $P = 1 - (1 - P^2)^{n-1}$. *When* $P > 1 - (1 - P^2)^{n-1}$ *the graph will tend not to be connected. When* $P < 1 - (1 - P^2)^{n-1}$ *the graph will tend be connected.*

Proof. For Erdös–Rényi graphs (Figure 49.42), all nondiagonal elements of the matrix have the same value p. Diagonal elements have the value zero. The formula $1 - (1 - P^2)^{n-1}$ follows directly from these two facts and Equation (49.26). When the value of this equation is equal to p, two nodes are just as likely to have a two-hop walk between them as a single edge. This means that connections of any number of hops are all equally likely. When the value of the equation is less than p, a walk of two hops is less probable than a single-hop connection. Since the equation is monotonically decreasing (increasing) as p decreases (increases), this means that longer walks are increasingly unlikely and the graph will tend not to be connected. By symmetry, when the value of the equation is greater than p the graph will tend to be connected.

49.18 Summary

Large-scale sensor networks will require the ability to organize themselves and adapt around unforeseen problems. Both of these requirements imply that their behavior will be at least partially nondeterministic. Our experience shows that mobile code and P2P networking are appropriate tools for implementing these systems.

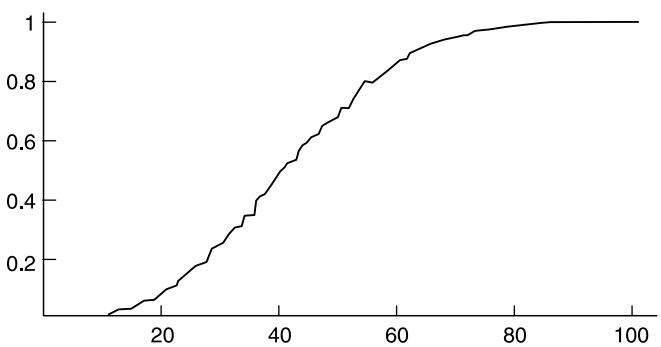


Figure 49.42. Empirical verification of theorem for Erdös–Rényi graph connectivity: 2000 instances of Erdös–Rényi graphs of five nodes were generated as the edge connection probability varied from 0.01 to 1.00. The x-axis times 0.01 is the edge probability. The y-axis is the percentage of graphs that were connected. The formula used in the theorem predicts the critical value around probability 0.4. When $p = 0.35$ (0.40) theorem (49.6) gives 0.357 (0.407).

© 2005 by Chapman & Hall/CRC

One difficulty with implementing adaptive infrastructures of this type is that it is difficult to estimate their performance. This chapter shows how random graph models can be used to estimate many important design parameters for these systems. They can also be used to quantify system performance.

Specifically, we have shown how to use random graph formalisms for wired and wireless P2P systems, such as those needed for sensor networks. Specifically, we have shown how to estimate:

- Network redundancy
- Expected number of hops
- System dependability
- QoS issues are handled Kapur *et al.* [9]
- System phase changes
- Vulnerability to intentional attack.

Acknowledgments and Disclaimer

This chapter is partially supported by the Office of Naval Research under Award No. N00014-01-1-0859 and by the Defense Advanced Research Projects Agency (DARPA) under ESP MURI Award No. DAAD19-01-1-0504 administered by the Army Research Office. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author and do not necessarily reflect the views of the Office of Naval Research (ONR), Defense Advanced Research Projects Agency (DARPA), and Army Research Office (ARO).

References

[1] Barabsi, A.-L., *Linked*, Perseus, Cambridge, MA, 2002.
[2] Watts, D.J., *Small Worlds*, Princeton University Press, Princeton, NJ, 1999.
[3] Albert, R. *et al.*, Error and attack tolerance of complex networks, *Nature*, 406, 378, 2000.
[4] Pastor-Storras, R. and Vespignani, A., Epidemic spreading in scale-free networks, *Physical Review Letters*, 86(14), 3200, 2001.
[5] Stauffer, D. and Aharony, A., *Introduction to Percolation Theory*, Taylor & Francis, London, 1994.
[6] Krishnamachari, B. *et al.*, Phase transition phenomena in wireless *ad-hoc* networks, in *Symposium on Ad-Hoc Wireless Networks, GlobeCom2001*, San Antonio, TX, November 2001, http://www.krishnamachari.net/papers/phaseTransitionWirelessNetworks.pdf (last accessed on 7/24/2004).
[7] Oram, A., *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, O'Reilly, Beijing, 2001.
[8] Lv, Q. *et al.*, Search and replication in unstructured peer-to-peer networks, in *International Conference on Supercomputing*, 2002, http://doi.acm.org/10.1145/514191.514206 (last accessed on 7/24/2004).
[9] Kapur, A. *et al.*, Design, performance and dependability of a peer-to-peer network supporting QoS for mobile code applications, in *Proceedings of the Tenth International Conference on Telecommunications Systems*, September 2002, 395.
[10] Aho, A.V. *et al.*, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.
[11] Cvetkovic, D.M., *et al.*, *Spectra of Graphs*, Academic Press, New York, 1979.
[12] Bollobás, B., *Random Graphs*, Cambridge University Press, Cambridge, UK, 2001.
[13] Albert, R. and Barabási, A.-L., Statistical mechanics of complex networks, arXiv:cond-mat/0106096v1, June 2001.
[14] Janson, S. *et al.*, *Random Graphs*, John Wiley, New York, 2000.
[15] Barabási, A.-L. and Albert, R., Emergence of scaling in random networks, *Science*, 286, 509, 512, 1999.

© 2005 by Chapman & Hall/CRC

[16] Press, W.H. *et al.*, *Numerical Recipes in FORTRAN*, Cambridge University Press, Cambridge, UK, 1992.
[17] Newman, M.E.J., Ego-centered networks and the ripple effect or why all your friends are weird, Working Papers, Santa Fe Institute, Santa Fe, NM, http://www.santafe.edu/sfi/publications/workingpapers/01-11-066.pdf (last accessed on 7/24/2004).
[18] Watts, D.J. *et al.*, personal correspondence.
[19] Reittu, H. and Norros, I., On the power law random graph model of Internet, submitted for review, 2002.
[20] Brooks, R.R. and Keiser, T., Mobile code daemons for networks of embedded systems, *IEEE Internet Computing*, 8(4), 72, 2004.
[21] Bonabeau, E. and Henaux, F., Graph partitioning with self-organizing maps, http://www.santafe.edu/sfi/publications/Abstracts/98-07-062abs.html (last accessed on 7/24/2004).
[22] Gu, M. *et al.*, Spectral relaxation methods and structure analysis for *K*-way graph clustering and bi-clustering, Technical Report, Department of Computer Science and Engineering, CSE-01-007, Pennsylvania State University, 2001.
[23] Dorogovtsev, S.N. and Mendez, J.F.F., Evolution of networks, ArXiv:cond-mat/0106096v1, June 2001.
[24] Newman, M.E.J. *et al.*, Random graphs with arbitrary degree distributions and their applications, arXiv:cond-mat/007235, May 7, 2001.

© 2005 by Chapman & Hall/CRC

50

On the Behavior of Communication Links in a Multi-Hop Mobile Environment

Prince Samar and Stephen B. Wicker

50.1 Introduction

In *ad hoc* and sensor networks the hardware for the network nodes is designed to be compact and lightweight to enable versatility and easy mobility. The transceivers of the nodes are thus constrained to run on limited power batteries. In order to conserve energy, nodes restrict their transmission power, allowing direct communication only with those nodes that are within their geographical proximity. To communicate with distant nodes in the network a node relies on multi-hop communication, whereby the source's data packets get forwarded along communication links between multiple pairs of nodes forming the route from the source to the destination.

As *ad hoc* and sensor networks do not require any pre-existing infrastructure and are self-organizing and self-configuring, they are amenable to a multitude of applications in diverse environments. These include battlefield deployments, where the transceivers may be mounted on unmanned aerial vehicles flying overhead, on moving armored vehicles, or may be carried by soldiers on foot. They may be used for communication during disaster-relief efforts or law enforcement operations in hostile environment. Such networks may be set up between students in a classroom or delegates at a convention center. Chemical, biological, or weather-related sensors may be spread around on land or on flotation devices at sea to monitor the environment and convey related statistics. Sensors may even be mounted on animals (e.g. whales, migratory birds, and other endangered species) to collect biological and environmental data.

With such a varied range of applications envisioned for *ad hoc* and sensor networks, the nodes in the network are expected to be mobile. Owing to limited transmission range, this implies that the set of communication links of a particular node may undergo frequent changes. These changes in the set of links of a node affect not only the node's ongoing communication, but also may impede the communication of other nodes due to the distributed, multi-hop nature of such networks.

© 2005 by Chapman & Hall/CRC

As the capacity and communication ability of *ad hoc* and sensor networks are dependent on the communication links [1], it is important to understand how the links of a node behave in a mobile environment. In this chapter we will analyze some of the important link properties of a node. The aim of the study is to gain an understanding of how the links behave and their properties vary depending on the network characteristics. The intuition developed can then be applied to design effective protocols for *ad hoc* and sensor networks.

The rest of the chapter is organized as follows. In Section 50.2 we discuss related work on characterizing the link behavior in an *ad hoc* or sensor network. Various properties of the links of a node in a mobile environment are derived in Section 50.3. In Section 50.4 we validate the derived expressions with simulation results. Section 50.5 discusses some applications of the derived properties and Section 50.6 concludes the chapter.

50.2 Related Work

Simulation has been the primary tool utilized in the literature to characterize and evaluate link properties in *ad hoc* and sensor networks. Some efforts have been directed at designing routing schemes that rely on identification of stable links in the network. Nodes make on-line measurements in order to categorize stable links, which are then preferentially used for routing. In associativity-based routing [2] nodes generate a beacon regularly to advertise their presence to their neighbors. A count of the number of beacons received from each neighbor is maintained in the form of associativity ''ticks'' which indicate the stability of a particular link. In signal-strength-based adaptive routing [3], received signal strength is also used in addition to location stability to quantify the reliability of a link. A routing metric is employed to select paths that consist of links with relatively strong signal strength and having an age above a certain threshold. Both of these approaches suffer from the fact that a link which is deemed stable based on past or current measurements may soon become unreliable compared with those currently categorized as unstable, due to the dynamic nature of mobile environments.

The route-lifetime assessment-based routing [4] uses an affinity parameter based on the measured rate of change of signal strength averaged over the last few samples in order to estimate the lifetime of a link. A metric combining the affinity parameter and the number of links in the route is then used to select routes for transmission control protocol traffic. However, shadow and multipath fading experienced by the received signal make the estimation of link lifetime very error prone. Su *et al.* [5] instead rely on information provided by a global positioning system about the current positions and velocities of two neighboring nodes to predict the expiration time of a link.

Empirical distributions of link lifetime and residual link lifetime have been presented [6] for different simulation parameters. Based on these results, two link stability metrics are also proposed to categorize stable links. The *edge effect* was identified by Lim *et al.* [7], which is the tendency of shortest routes in high-density wireless networks to be unstable. This is because such routes are usually composed of nodes that lie at the edges of each others' transmission ranges, so that a relatively small movement of any node in the route is sufficient to break it. Estimated stability of links has been used as the basis of route caching strategies for reactive routing protocols [8].

Analytical studies of link properties in a mobile network have been limited, partly due to the abstruse nature of the problem. Though a number of mobility models have been proposed and used in the literature [9], none of them is satisfactory for representing node mobility in general. The expected link lifetime of a node is examined for some simple mobility scenarios by Turgut *et al.* [10]. It is shown that the expected link lifetime under Brownian motion is infinite, while under deterministic mobility it can be found explicitly, given the various parameters.

A random mobility model was developed by McDonald and Znati [11] and then used to quantify the probability that a link will be available between two nodes after an interval of duration t, given that the link exists between them at time t_0. This probability is then used to evaluate the availability of a path after a duration t, assuming independent link failures. This forms the basis of a dynamic clustering algorithm such that more reliable members get selected to form the cluster. However, selection of paths

© 2005 by Chapman & Hall/CRC

for routing using this criterion may not be practical, as the model considers a link to be available at time $t_0 + t$ even when it undergoes failure during one or more intervals between t_0 and $t_0 + t$. When a link of a route actively being used breaks it may be necessary to find an alternate route immediately, instead of just waiting indefinitely for the link to become available again. Jiang *et al.* [12] tried to overcome this drawback by estimating the probability that a link between two nodes will be continuously available for a period T_p, where T_p is predicted based on the nodes' current movements.

A number of issues related to the behavior of links still remain unexplored. In this chapter we develop an analytical framework in order to investigate some important characteristics of the links of a node in a mobile environment. The derived link properties can be instrumental in the design and analysis of networking algorithms, as illustrated by the discussion on the few example applications in Section 50.5.

50.3 Link Properties

Here, we derive analytical expressions for a number of link properties: (a) expected lifetime of a link, (b) probability distribution of link lifetime, (c) expected rate of new link arrivals, (d) probability distribution of new link interarrival time, (e) expected rate of link change, (f) probability distribution of link breakage interarrival time, (g) probability distribution of link change interarrival time, and (h) expected number of neighbors. These expressions will help us understand better the behavior of these properties and their dependence on various network parameters.

In order to model the network for the analyses, we make the following assumptions:

1. A node has a bidirectional communication link with any other node within a distance of R meters from it. The link breaks if the node moves to a distance greater than R.
2. A node in the network moves with a constant velocity which is uniformly distributed between a meters/second and b meters/second.
3. The direction of a node's velocity is uniformly distributed between 0 and 2π.
4. A node's speed, its direction of motion, and its location are uncorrelated.
5. The locations of nodes in the network are modeled by a two-dimensional Poisson point process with intensity σ such that for a network region $\mathbf{D}$ with an area A, the probability that $\mathbf{D}$ contains k nodes is given by

$$\text{Prob}(k \text{ nodes in } \mathbf{D}) = \frac{(\sigma A)^k \, e^{-\sigma A}}{k!} \tag{50.1}$$

Assumption 1 implies that the signal-to-interference ratio (SIR) remains high up to a certain distance R from the transmitter, enabling nearly perfect estimation of the transmitted signal. However, SIR drops beyond this distance, rapidly increasing the bit error rate to unacceptable levels. Though the shadow and multipath fading experienced by the received signal may make the actual transmission zone unsymmetrical, this is a fair approximation if all the nodes in the network use the same transmission power. This simplifying assumption is commonly used in the simulation and analysis of *ad hoc* and sensor networks.

Assumption 2 models a mobile environment where nodes are moving around with different velocities that are uniformly distributed between two limits. This high mobility model is chosen as it is challenging for network communication and can, thus, facilitate finding "worst-case" bounds on the link properties for a general scenario. It is to be noted that the intensity of mobility being modeled can be changed by appropriately choosing the two parameters, a and b.

Assumptions 2–5 characterize the aggregate behavior of nodes in a large network. Owing to the large number of independent nodes operating in an *ad hoc* fashion, any correlation between nodes can be assumed to be insignificant. Although it is possible that some nodes may share similar objectives and may move together, a large enough population of autonomous nodes can be expected in the network so that the composite effect can be modeled by a random process.

© 2005 by Chapman & Hall/CRC

Assumption 5 indicates the location distribution of nodes in the network at any time. Poisson processes model "total randomness," thus reflecting the randomness shown by the aggregate behavior of nodes in a large network. This assumption is frequently used to model the location of nodes in an *ad hoc* or cellular network. Using Equation (50.1), it is easy to see that the expected number of nodes in **D** is equal to σA. Thus, σ represents the average density of nodes in the network.

50.3.1 Expected Link Lifetime

Figure 50.1 shows the transmission zone of a node (say node 1) which is a circle of radius R centered at the node. The figure shows the trajectory of another node (say node 2) entering the transmission zone of node 1 at A, traveling along AB, and exiting the transmission zone at B.

With respect to a stationary Cartesian coordinate system with orthogonal unit vectors $\hat{i}$ and $\hat{j}$ along the X and Y axes respectively, let the velocity of node 1 be $\vec{v}_1 = v_1\hat{i}$ and the velocity of node 2, which makes an angle θ with the positive X axis, be $\vec{v}_2 = v_2 \cos\theta\hat{i} + v_2 \sin\theta\hat{j}$. Hence, the relative velocity of node 2 with respect to node 1 is

$$\vec{v} \triangleq \vec{v}_{21} = \vec{v}_2 - \vec{v}_1 = (v_2\cos\theta - v_1)\hat{i} + v_2\sin\theta\hat{j} \tag{50.2}$$

Consider a Cartesian coordinate system $X'Y'$ fixed on node 1 such that the X' and Y' axes are parallel to $\hat{i}$ and $\hat{j}$ respectively, as shown in Figure 50.1. The magnitude of node 2's velocity in this coordinate system is

$$v \triangleq |\vec{v}| = \sqrt{v_1^2 + v_2^2 - 2v_1v_2\cos\theta} \tag{50.3}$$

and its direction of motion in this coordinate system, as indicated in Figure 50.1, is

$$\phi \triangleq \angle\vec{v} = \tan^{-1}\left(\frac{\sin\theta}{\cos\theta - v_1/v_2}\right) \tag{50.4}$$

Let the point of entry A of node 2 in node 1's transmission zone be defined by an angle α, measured clockwise from OX''. Thus, point A has coordinates $(-R\cos\alpha, R\sin\alpha)$ in the $X'Y'$ coordinate system. In Figure 50.1, $OA = OB = R$. AB makes an angle ϕ with the horizontal, which is the direction of the

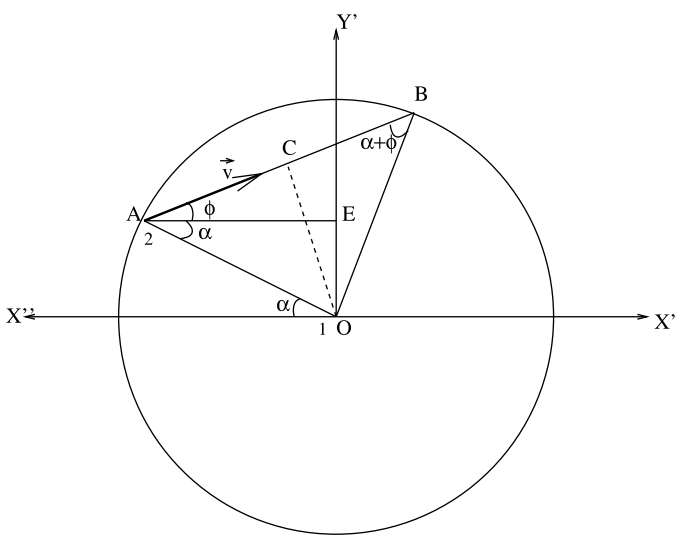


Figure 50.1. The transmission zone of node 1 at O with node 2 entering the zone at A and exiting at B.

© 2005 by Chapman & Hall/CRC

relative velocity of node 2. Line OC is perpendicular to AB. As OAB makes an isosceles triangle, $\angle OAB = \angle OBA = \alpha + \phi$. Therefore, $AC = BC = R\cos(\alpha + \phi)$. As θ and ϕ can have any value between 0 and 2π, the distance d_{link} that node 2 travels inside node 1's zone is

$$d_{\text{link}} = |2R\cos(\alpha + \phi)| = 2R|\cos(\alpha + \phi)| \tag{50.5}$$

Hence, the time that node 2 spends inside node 1's zone, which is equal to the time for which the link between node 1 and node 2 remains active, is

$$\begin{aligned} t_{\text{link}} &= \frac{d_{\text{link}}}{|\vec{v}|} \\ &= \frac{2R|\cos(\alpha + \phi)|}{v} \end{aligned} \tag{50.6}$$

The average link lifetime can be calculated as the expectation of t_{link} over v, ϕ, α.

$$\bar{T}_{\text{link}}(v_1) = E_{v\phi\alpha}\left[t_{\text{link}}(v, \phi, \alpha)\right] \tag{50.7}$$

Let the joint probability density function (PDF) of v, ϕ, α for nodes that enter the zone be $f_{v\phi\alpha}(v, \phi, \alpha)$. It can be expressed as

$$f_{v\phi\alpha}(v, \phi, \alpha) = f_{\alpha|v\phi}(\alpha|v, \phi) f_{v\phi}(v, \phi) \tag{50.8}$$

where $f_{\alpha|v\phi}(\alpha|v, \phi)$ is the conditional probability density of α given the relative velocity $\vec{v}$; and $f_{v\phi}(v, \phi)$ is the joint probability density of the magnitude v and phase ϕ of $\vec{v}$. Expressions for these probability density functions are derived in Appendix 50A.

Thus, the expected link lifetime can be calculated as

$$\begin{aligned} \bar{T}_{\text{link}}(v_1) &= \int_{v=0}^{\infty} \int_{\phi=-\pi}^{\pi} \int_{\alpha=-\pi}^{\pi} t_{\text{link}} f_{v\phi\alpha}(v, \phi, \alpha)\, \mathrm{d}\alpha\, \mathrm{d}\phi\, \mathrm{d}v \\ &= \int_0^{\infty} \int_{-\pi}^{\pi} f_{v\phi}(v, \phi) \left\{ \int_{-\pi}^{\pi} \frac{2R|\cos(\alpha + \phi)|}{v} f_{\alpha|v\phi}(\alpha|v, \phi)\, \mathrm{d}\alpha \right\} \mathrm{d}\phi\, \mathrm{d}v \\ &= \frac{R}{2(b-a)} \int_0^{\infty} \int_0^{\pi} \frac{1}{\sqrt{v^2 + v_1^2 + 2vv_1\cos\phi}} \left\{ u\left[\sqrt{v^2 + v_1^2 + 2vv_1\cos\phi} - a\right] \right. \\ &\quad \left. - u\left[\sqrt{v^2 + v_1^2 + 2vv_1\cos\phi} - b\right] \right\} \mathrm{d}v\, \mathrm{d}\phi \end{aligned} \tag{50.9}$$

In order to eliminate the unit step function $u(\cdot)$ from the integral in Equation (50.9), one needs to identify the values of v which satisfy the following two inequalities:

$$\sqrt{v^2 + v_1^2 + 2vv_1\cos\phi} - a \geq 0 \tag{50.10}$$

$$\sqrt{v^2 + v_1^2 + 2vv_1\cos\phi} - b < 0 \tag{50.11}$$

© 2005 by Chapman & Hall/CRC

The range of $v \geq 0$ satisfying Equations (50.10) and (50.11) are:

- $v \in \left[0,\ -v_1 \cos\phi + \sqrt{b^2 - v_1^2 \sin^2\phi}\right]$ if $0 \leq \phi \leq \left[\pi - \sin^{-1}(a/v_1)\right]$
- $v \in \left[0,\ -v_1 \cos\phi - \sqrt{a^2 - v_1^2 \sin^2\phi}\right] \bigcup \left[-v_1 \cos\phi + \sqrt{a^2 - v_1^2 \sin^2\phi},\ -v_1 \cos\phi + \sqrt{b^2 - v_1^2 \sin^2\phi}\right]$ if $\left[\pi - \sin^{-1}(a/v_1)\right] \leq \phi \leq \pi$

Hence:

$$\bar{T}_{\text{link}}(v_1) = \frac{R}{2(b-a)}\left[\int_0^{\pi - \sin^{-1}(a/v_1)} \int_0^{-v_1\cos\phi + \sqrt{b^2 - v_1^2\sin^2\phi}} \frac{1}{\sqrt{v^2 + v_1^2 + 2vv_1\cos\phi}}\,dv\,d\phi \right.$$
$$+ \int_{\pi - \sin^{-1}(a/v_1)}^{\pi} \left(\int_0^{-v_1\cos\phi - \sqrt{a^2 - v_1^2\sin^2\phi}} \frac{1}{\sqrt{v^2 + v_1^2 + 2vv_1\cos\phi}}\,dv \right.$$
$$\left.\left. + \int_{-v_1\cos\phi + \sqrt{a^2 - v_1^2\sin^2\phi}}^{-v_1\cos\phi + \sqrt{b^2 - v_1^2\sin^2\phi}} \frac{1}{\sqrt{v^2 + v_1^2 + 2vv_1\cos\phi}}\,dv \right) d\phi \right] \tag{50.12}$$

Equation (50.12) can be simplified to give

$$\bar{T}_{\text{link}}(v_1) = \frac{R}{2(b-a)}\left(\int_0^{\pi} \log\left| \frac{b + \sqrt{b^2 - v_1^2\sin^2\phi}}{v_1 + v_1\cos\phi} \right| d\phi \right.$$
$$\left. - \int_{\pi - sin^{-1}(a/v_1)}^{\pi} \log\left| \frac{a + \sqrt{a^2 - v_1^2\sin^2\phi}}{a - \sqrt{a^2 - v_1^2\sin^2\phi}} \right| d\phi \right) \tag{50.13}$$

In particular, if $a = 0$ then the above expression reduces to

$$\bar{T}_{\text{link}}(v_1) = \frac{R}{2b}\left(\int_0^{\pi} \log\left| \frac{b + \sqrt{b^2 - v_1^2\sin^2\phi}}{v_1 + v_1\cos\phi} \right| d\phi \right) \tag{50.14}$$

Equation (50.13) cannot be integrated into an explicit function. However, it can be numerically integrated to give the expected link lifetime for the chosen distribution of mobility in the network.

Figure 50.2 plots the expected link lifetime for a node as a function of its velocity. The velocity of the nodes in the network is assumed to be uniformly distributed between [0, 40] m/s. As can be observed from the plot, the expected link lifetime for a node decreases rapidly as its velocity is increased. As an illustration, links last almost three times longer, on average, for a node moving with a velocity of 5 m/s compared with a node moving with a velocity of 40 m/s. Also, as can be seen from Equation (50.13), the expected link lifetime is directly proportional to the transmission radius R of a node.

It is to be noted that Assumption 5 was not needed for determining the expected link lifetime and, thus, the derived expression is independent of the density of nodes in the network. This is because $\bar{T}_{\text{link}}(v_1)$ is averaged over link lifetimes corresponding to the range of velocities present in the network weighted by their probability density, without regard to how many or how often these links are formed.

© 2005 by Chapman & Hall/CRC

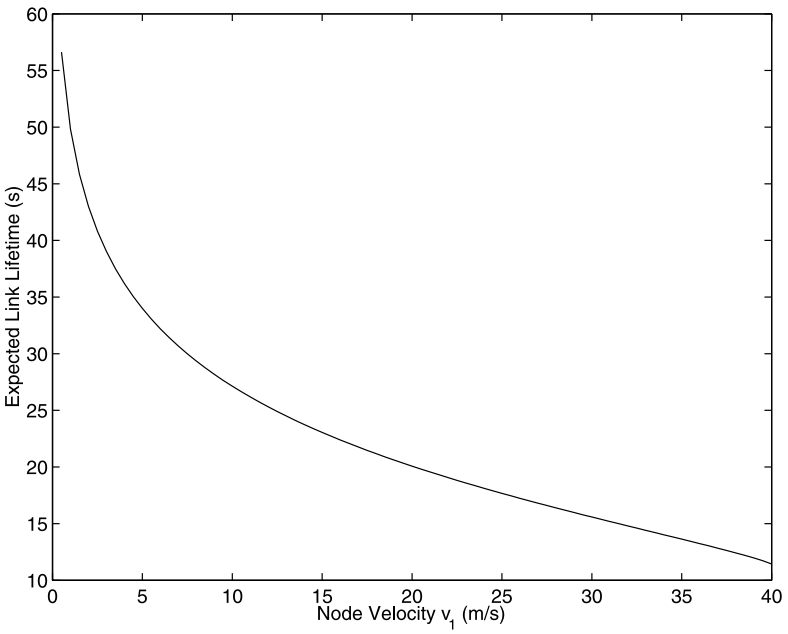


Figure 50.2. Expected link lifetime of a node as a function of its velocity, where $a = 0$ m/s, $b = 40$ m/s and $R = 250$ m.

50.3.2 Link Lifetime Distribution

For a particular node moving with a velocity v_1, the cumulative distribution function (CDF) of the link lifetime is given by

$$F_{\text{link}}^{v_1}(t) = \text{Prob}\{t_{\text{link}} \leq t\} \tag{50.15}$$

Clearly, $F_{\text{link}}^{v_1}(t) = 0$ for $t < 0$. For $t \geq 0$, we have

$$\begin{aligned} F_{\text{link}}^{v_1}(t) &= \text{Prob}\left\{ \frac{2R|\cos(\alpha+\phi)|}{v} \leq t \right\} \\ &= 1 - \text{Prob}\left\{ \frac{2R|\cos(\alpha+\phi)|}{v} > t \right\} \\ &= 1 - \text{Prob}\left\{ |\cos(\alpha+\phi)| > \frac{vt}{2R} \right\} \end{aligned} \tag{50.16}$$

Now:

$$\begin{aligned} \text{Prob}\left\{ |\cos(\alpha+\phi)| > \frac{vt}{2R} \right\} &= \text{Prob}\left\{ -\cos^{-1}\left(\frac{vt}{2R}\right) - \phi \leq \alpha \leq \cos^{-1}\left(\frac{vt}{2R}\right) - \phi, v \leq \frac{2R}{t} \right\} \\ &= \int_{\phi=-\pi}^{\pi} \int_{v=0}^{2R/t} \int_{\alpha=-\cos^{-1}(vt/2R)-\phi}^{\cos^{-1}(vt/2R)-\phi} f_{v\phi\alpha}(v,\, \phi,\, \alpha)\, \mathrm{d}\alpha\, \mathrm{d}v\, \mathrm{d}\phi \\ &= \int_{\phi=-\pi}^{\pi} \int_{v=0}^{2R/t} f_{v\phi}(v,\, \phi) \left(\int_{-\cos^{-1}(vt/2R)-\phi}^{\cos^{-1}(vt/2R)-\phi} f_{\alpha|v\phi}(\alpha|v,\phi)\, \mathrm{d}\alpha \right) \mathrm{d}v\, \mathrm{d}\phi \end{aligned} \tag{50.17}$$

© 2005 by Chapman & Hall/CRC

Using the expression of $f_{\alpha|v\phi}(\alpha|v, \phi)$ from Equation (50A.3), Equation (50.17) can be simplified to give

$$\begin{aligned}\text{Prob}\left\{|\cos(\alpha+\phi)| > \frac{vt}{2R}\right\} &= \int_{\phi=-\pi}^{\pi}\int_{v=0}^{2R/t} f_{v\phi}(v,\phi)\sqrt{1-\left(\frac{vt}{2R}\right)^2}\,\mathrm{d}v\,\mathrm{d}\phi \\ &= \frac{1}{\pi(b-a)}\int_0^{\pi}\int_0^{2R/t}\frac{v}{\sqrt{v^2+v_1^2+2vv_1\cos\phi}} \\ &\quad\times\sqrt{1-\left(\frac{vt}{2R}\right)^2}\left[u\left(\sqrt{v^2+v_1^2+2vv_1\cos\phi}-a\right)\right. \\ &\quad\left.-u\left(\sqrt{v^2+v_1^2+2vv_1\cos\phi}-b\right)\right]\mathrm{d}v\,\mathrm{d}\phi \end{aligned} \tag{50.18}$$

Substituting in Equation (50.16), we get an expression for the CDF of the link lifetime of a node moving with a velocity v_1:

$$\begin{aligned}F_{\text{link}}^{v_1}(t) = 1 - \frac{1}{\pi(b-a)}\int_0^{\pi}\int_0^{2R/t}\frac{v}{\sqrt{v^2+v_1^2+2vv_1\cos\phi}}\sqrt{1-\left(\frac{vt}{2R}\right)^2} \\ \left[u\left(\sqrt{v^2+v_1^2+2vv_1\cos\phi}-a\right)-u\left(\sqrt{v^2+v_1^2+2vv_1\cos\phi}-b\right)\right]\mathrm{d}v\,\mathrm{d}\phi\end{aligned} \tag{50.19}$$

No closed-form solution for the integrals in Equation (50.19) exists. However, Equation (50.19) can be numerically integrated to give the CDF of the link lifetime of a node moving with velocity v_1. Figure 50.3 plots the link lifetime CDF for different node velocities v_1, where $a = 0$ m/s, $b = 40$ m/s and $R = 250$ m.

The PDF $f_{\text{link}}^{v_1}(t)$ of link lifetime is found by differentiating Equation (50.19) with respect to t. Figure 50.4 plots the PDF by numerically differentiating the curves in Figure 50.3. Note that, for $v_1 > 0$,

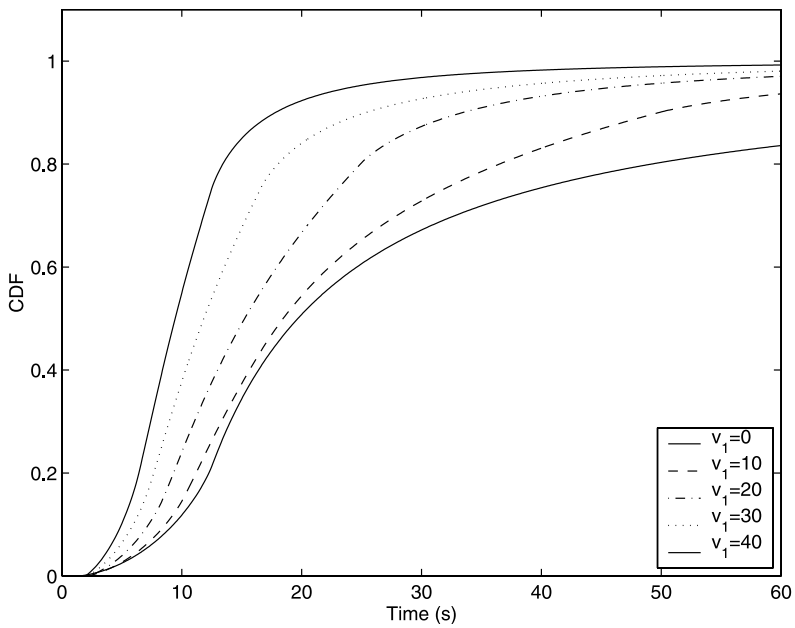


Figure 50.3. The CDF of the link lifetime of a node moving with velocity v_1, for $a = 0$ m/s, $b = 40$ m/s and $R = 250$ m.

© 2005 by Chapman & Hall/CRC

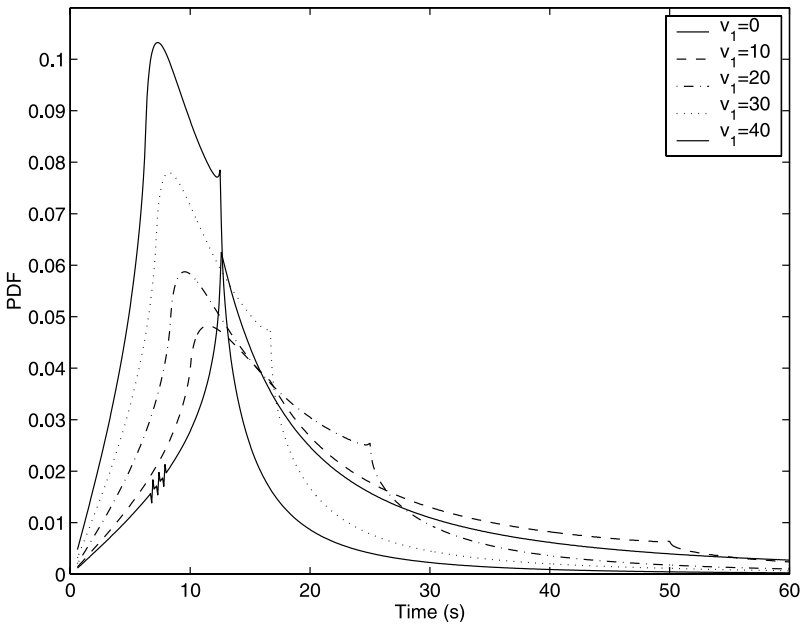


Figure 50.4. The PDF of the link lifetime of a node moving with velocity v_1, for $a = 0$ m/s, $b = 40$ m/s and $R = 250$ m.

the point where the PDF curve is not differentiable corresponds to $t = 2R/v_1$. Also, it can be seen that the maxima of the PDF curve, which correspond to the *mode*s of the distribution, shift towards the left as the node velocity increases.

As in Section 50.3.1, the expression derived does not depend on the density or location distribution of nodes in the network.

50.3.3 Expected New Link Arrival Rate

Consider Figure 50.5, which shows the transmission zone of node 1 moving with velocity $\vec{v}_1$ with respect to the stationary coordinate system XY, as defined before. For given values of v and ϕ, any node with relative velocity $\vec{v} = v\cos\phi\hat{i} + v\sin\phi\hat{j}$ with respect to node 1 can only enter node 1's transmission zone from a point on the semi-circle $\alpha \in [-(\frac{\pi}{2} + \phi), \frac{\pi}{2} - \phi]$,[1] as seen in Appendix 50A. Thus, a node with relative velocity $\vec{v}$ would enter the transmission zone within the next t seconds if it is currently located in the shaded region $\mathbf{D_a}$ of Figure 50.5, which is composed of all points at most vt meters away measured along angle ϕ from the semicircle $\alpha \in [-(\frac{\pi}{2} + \phi), \frac{\pi}{2} - \phi]$.

The area of the shaded region $\mathbf{D_a}$ is $A = vt2R$. Using Assumption 5, the average number of nodes in $\mathbf{D_a}$ is found to be equal to $2Rvt\sigma$. The average number of nodes in $\mathbf{D_a}$ with velocity $\vec{v}$ is equal to $2R\sigma vtf(v, \phi)\,\mathrm{d}v\,\mathrm{d}\phi$. This is just the average number of nodes with velocity $\vec{v}$ entering the zone within the next t seconds. The total expected number of nodes entering the zone within the next t seconds is found by integrating this quantity over all possible values of v and ϕ.

$$E\{\text{number of nodes entering the zone in } t \text{ seconds} = \eta(v_1) = \int_{v=0}^{\infty}\int_{\phi=-\pi}^{\pi} 2R\sigma vtf(v, \phi)\,\mathrm{d}v\,\mathrm{d}\phi \quad (50.20)$$

[1] α, as defined before, is the angle measured clockwise from the negative X axis of the coordinate system fixed on node 1.

© 2005 by Chapman & Hall/CRC

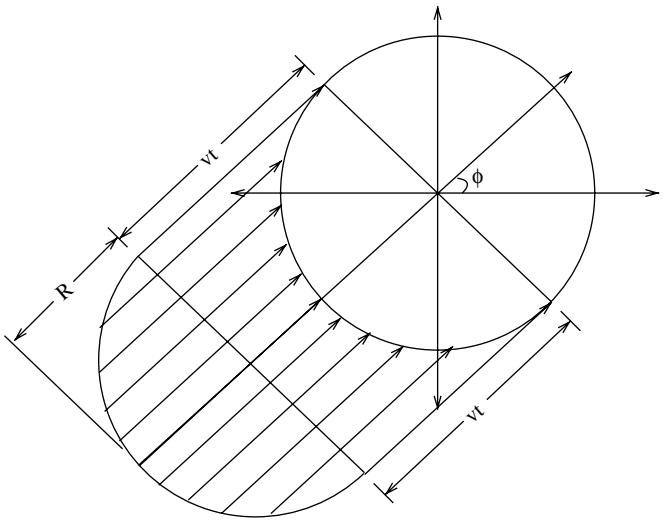


Figure 50.5. Calculation of expected new link arrival rate.

where $f(v, \phi)$, the joint probability density of a node's relative velocity, has been derived in Equation (50A.13). Thus Equation (50.20) can be expressed as

$$\begin{aligned}\eta(v_1) &= \frac{R\sigma t}{\pi(b-a)} \int_{-\pi}^{\pi} \int_0^{\infty} \frac{v^2}{\sqrt{v^2 + v_1^2 + 2vv_1 \cos\phi}} \left[u\left(\sqrt{v^2 + v_1^2 + 2vv_1 \cos\phi} - a\right) \right. \\ &\qquad \left. - u\left(\sqrt{v^2 + v_1^2 + 2vv_1 \cos\phi} - b\right) \right] \mathrm{d}v \, \mathrm{d}\phi \\ &= \frac{2R\sigma t}{\pi(b-a)} \left(\int_0^{\pi} \int_0^{-v_1 \cos\phi + \sqrt{b^2 - v_1^2 \sin^2\phi}} \frac{v^2}{\sqrt{v^2 + v_1^2 + 2vv_1 \cos\phi}} \, \mathrm{d}v \, \mathrm{d}\phi \right. \\ &\qquad \left. - \int_{\pi - \sin^{-1}(a/v_1)}^{\pi} \int_{-v_1 \cos\phi - \sqrt{a^2 - v_1^2 \sin^2\phi}}^{-v_1 \cos\phi + \sqrt{a^2 - v_1^2 \sin^2\phi}} \frac{v^2}{\sqrt{v^2 + v_1^2 + 2vv_1 \cos\phi}} \, \mathrm{d}v \, \mathrm{d}\phi \right) \end{aligned} \tag{50.21}$$

The above can be simplified to give

$$\begin{aligned}\eta(v_1) &= \frac{2R\sigma t}{\pi(b-a)} \left\{ b^2 \mathcal{E}\left(\frac{v_1}{b}\right) - 2a^2 \mathcal{E}\left(\frac{v_1}{a}\right) + a^2 \mathcal{E}\left[\pi - \sin^{-1}\left(\frac{a}{v_1}\right), \frac{v_1}{a}\right] \right. \\ &\qquad + \frac{v_1^2}{4} \int_0^{\pi} [1 + 3\cos(2\phi)] \log \left| \frac{b + \sqrt{b^2 - v_1^2 \sin^2\phi}}{v_1 + v_1 \cos\phi} \right| \mathrm{d}\phi \\ &\qquad \left. - \frac{v_1^2}{4} \int_{\pi - \sin^{-1}(a/v_1)}^{\pi} [1 + 3\cos(2\phi)] \log \left| \frac{a + \sqrt{a^2 - v_1^2 \sin^2\phi}}{a - \sqrt{a^2 - v_1^2 \sin^2\phi}} \right| \mathrm{d}\phi \right\} \end{aligned} \tag{50.22}$$

where $\mathcal{E}(\cdot)$ is the complete elliptic integral of the second kind and $\mathcal{E}(\cdot, \cdot)$ is the incomplete elliptic integral of the second kind.

© 2005 by Chapman & Hall/CRC

Thus, the expected number of nodes entering the transmission zone per second or, equivalently, the rate of new link arrivals, is given by

$$\dot{\eta}(v_1) = \frac{2R\sigma}{\pi(b-a)} \left\{ b^2 \mathcal{E}\left(\frac{v_1}{b}\right) - 2a^2 \mathcal{E}\left(\frac{v_1}{a}\right) + a^2 \mathcal{E}\left(\pi - \sin^{-1}\left(\frac{a}{v_1}\right),\ \frac{v_1}{a}\right) \right.$$
$$+ \frac{v_1^2}{4} \int_0^{\pi} [1 + 3\cos(2\phi)] \log \left| \frac{b + \sqrt{b^2 - v_1^2 \sin^2\phi}}{v_1 + v_1 \cos\phi} \right| \mathrm{d}\phi$$
$$\left. - \frac{v_1^2}{4} \int_{\pi - \sin^{-1}(a/v_1)}^{\pi} [1 + 3\cos(2\phi)] \log \left| \frac{a + \sqrt{a^2 - v_1^2 \sin^2\phi}}{a - \sqrt{a^2 - v_1^2 \sin^2\phi}} \right| \mathrm{d}\phi \right\} \tag{50.23}$$

When $a = 0$, $\dot{\eta}(v_1)$ reduces to

$$\dot{\eta}(v_1) = \frac{2R\sigma}{\pi b} \left\{ b^2 \mathcal{E}\left(\frac{v_1}{b}\right) + \frac{v_1^2}{4} \int_0^{\pi} [1 + 3\cos(2\phi)] \log \left| \frac{b + \sqrt{b^2 - v_1^2 \sin^2\phi}}{v_1 + v_1 \cos\phi} \right| \mathrm{d}\phi \right\} \tag{50.24}$$

In Figure 50.6 we plot the expected rate of new link arrivals for a node moving with velocity v_1. While generating the curves, the values of the parameters are set to $a = 0$ m/s, $b = 40$ m/s, $R = 250$ m and $\sigma = \rho/\pi R^2$ nodes/m^2. Note that ρ represents the average number of nodes within a transmission zone.

An important point to observe from Equation (50.23) is that the expected rate of new link arrivals for a node is directly proportional to the average density σ of nodes in the network. It is also directly proportional to the transmission radius R of a node.

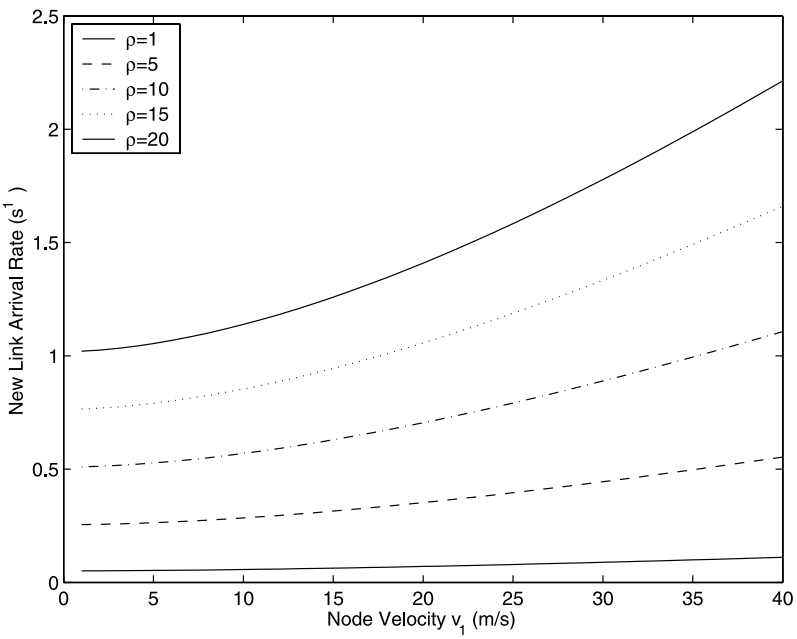


Figure 50.6. Rate of new link arrivals for a node moving with velocity v_1, where $a = 0$ m/s, $b = 40$ m/s, $R = 250$ m and $\sigma = \rho/\pi R^2$ nodes/m^2.

© 2005 by Chapman & Hall/CRC

50.3.4 New Link Interarrival Time Distribution

The cumulative distribution function of new link interarrival time is given by

$$F^{v_1}_{\text{arrival}}(t) = \text{Prob}\ \{\text{link interarrival time} \leq t\} \tag{50.25}$$

$\mathbf{D_a}$, the shaded region of Figure 50.5, has an area $A = 2Rvt$. As seen in Section 50.3.2, a node with velocity $\vec{v} = v\cos\phi\hat{i} + v\sin\phi\hat{j}$ currently located in $\mathbf{D_a}$ will enter the transmission zone within the next t seconds. Thus, given $\vec{v}$, the probability that the link interarrival time is not more than t is equal to the probability that there exists at least one node in $\mathbf{D_a}$ with velocity $\vec{v}$. Therefore, using Assumption 5:

$$\begin{aligned}\text{Prob}\{\text{link interarrival time} \leq t|v, \phi\} &= \text{Prob}\{\text{at least 1 node in } \mathbf{D_a}|\mathbf{v}, \phi\} \\ &= 1 - \text{Prob}\{\text{no node in } \mathbf{D_a}|\mathbf{v}, \phi\} \\ &= 1 - e^{-\sigma A} \\ &= 1 - e^{-2R\sigma tv}\end{aligned} \tag{50.26}$$

Hence, the CDF of new link interarrival time can be expressed as

$$F^{v_1}_{\text{arrival}}(t) = \iint_{v,\phi} \left(1 - e^{-2R\sigma tv}\right) f(v, \phi)\, dv\, d\phi \tag{50.27}$$

Substituting for $f(v, \phi)$ from Equation (50A.13):

$$\begin{aligned}F^{v_1}_{\text{arrival}}(t) = 1 - \frac{1}{\pi(b-a)} \int_0^{\pi} \int_0^{\infty} e^{-2R\sigma tv} \frac{v}{\sqrt{v^2 + v_1^2 + 2vv_1\cos\phi}} \\ \left[u\left(\sqrt{v^2 + v_1^2 + 2vv_1\cos\phi} - a\right) - u\left(\sqrt{v^2 + v_1^2 + 2vv_1\cos\phi} - b\right)\right] dv\, d\phi\end{aligned} \tag{50.28}$$

The PDF $f^{v_1}_{\text{arrival}}(t)$ of new link interarrival time is given by

$$\begin{aligned}f^{v_1}_{\text{arrival}}(t) &= \frac{d}{dt} F^{v_1}_{\text{arrival}}(t) \\ &= \frac{2R\sigma}{\pi(b-a)} \int_0^{\pi} \int_0^{\infty} e^{-2R\sigma tv} \frac{v^2}{\sqrt{v^2 + v_1^2 + 2vv_1\cos\phi}} \\ &\quad \left[u\left(\sqrt{v^2 + v_1^2 + 2vv_1\cos\phi} - a\right) - u\left(\sqrt{v^2 + v_1^2 + 2vv_1\cos\phi} - b\right)\right] dv\, d\phi\end{aligned} \tag{50.29}$$

Figure 50.7 illustrates the new link interarrival time distribution for a node moving with velocity v_1, for $a = 0$ m/s, $b = 40$ m/s, $R = 250$ m and $\sigma = 10/\pi R^2$ nodes/m^2. The corresponding new link interarrival time density for different node velocities v_1 is plotted in Figure 50.8. It can be observed that the new link interarrival time PDF curves drop rapidly as time t increases.

50.3.5 Expected Link Change Rate

Any change in the set of links of a node may be either due to the arrival of a new link or due to the breaking of a currently active link. Thus, the expected link change rate for a node is equal to the sum of the expected new link arrival rate and the expected link breakage rate. The expected new link arrival rate has been found earlier; see Equation (50.23).

© 2005 by Chapman & Hall/CRC

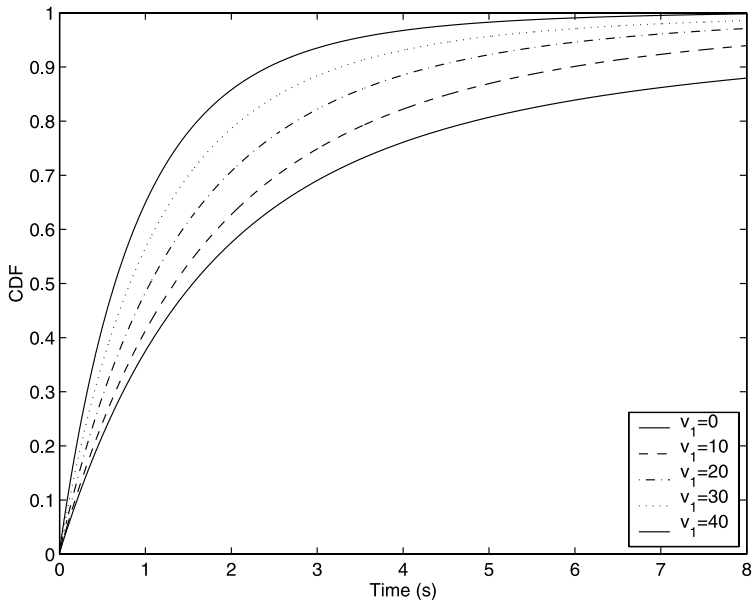


Figure 50.7. The CDF of new link interarrival time for a node moving with velocity v_1, for $a = 0$ m/s, $b = 40$ m/s, $R = 250$ m and $\sigma = 10/\pi R^2$ nodes/m^2.

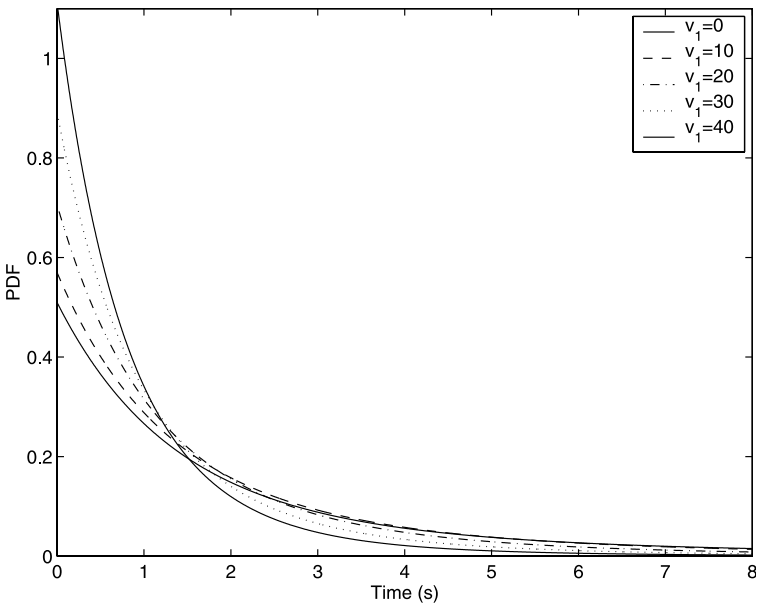


Figure 50.8. The PDF of new link interarrival time for a node moving with velocity v_1, for $a = 0$ m/s, $b = 40$ m/s, $R = 250$ m and $\sigma = 10/\pi R^2$ nodes/m^2.

In order to determine the expected link breakage rate, suppose that the network is formed at time $t = 0$. Let the total number of new link arrivals for a node between $t = 0$ and $t = t_0$ be $\eta(t_0)$ and the total number of link breakages for the node during the same interval be $\mu(t_0)$. Let the number of neighbors of the node at time $t = t_0$ be $N(t_0)$. Now:

$$\eta(t_0) - \mu(t_0) = N(t_0) \tag{50.30}$$

© 2005 by Chapman & Hall/CRC

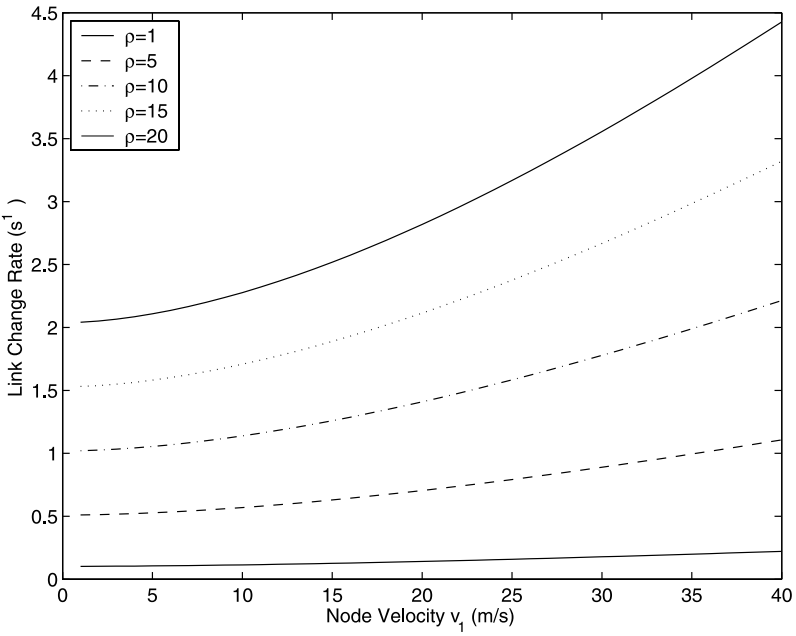


Figure 50.9. Expected link change arrival rate for a node moving with velocity v_1, where $a = 0$ m/s, $b = 40$ m/s, $R = 250$ m and $\sigma = \rho/\pi R^2$ nodes/m^2.

Dividing both the sides in Equation (50.30) by t_0:

$$\frac{\eta(t_0)}{t_0} - \frac{\mu(t_0)}{t_0} = \frac{N(t_0)}{t_0} \tag{50.31}$$

Taking the limit as $t \to \infty$ in Equation (50.31), $\eta(t_0)/t_0$ equals the expected rate of new link arrivals $\dot{\eta}$ and $\mu(t_0)/t_0$ equals the expected rate of link breakages $\dot{\mu}$ (assuming ergodicity). If the number of neighbors of a node is bounded,[2] $N(t_0)/t_0 \to 0$ as $t \to \infty$. This implies that $\dot{\mu} = \dot{\eta}$, i.e. the expected rate of link breakages is equal to the expected rate of new link arrivals. Thus, the expected link change arrival rate $\dot{\gamma}(v_1)$ for a node moving with velocity v_1 is given by

$$\begin{aligned} \dot{\gamma}(v_1) &= \dot{\eta}(v_1) + \dot{\mu}(v_1) \\ &= 2\dot{\eta}(v_1) \end{aligned} \tag{50.32}$$

where $\dot{\eta}(v_1)$ is as expressed in Equation (50.23).

The expected link change arrival rate as a function of the node velocity v_1 is plotted in Figure 50.9, where $a = 0$ m/s, $b = 40$ m/s, $R = 250$ m and $\sigma = \rho/\pi R^2$ nodes/m^2. Like $\dot{\eta}(v_1)$, $\dot{\gamma}(v_1)$ is also directly proportional to the average node density σ and the node transmission radius R.

50.3.6 Link Breakage Interarrival Time Distribution

In order to derive the link breakage interarrival time distribution, we proceed in a manner similar to Section 50.3.4. Consider Figure 50.10 showing the transmission zone of node 1. The shaded region $\mathbf{D_b}$

[2]Which is the case for any practical *ad hoc* or sensor network.

© 2005 by Chapman & Hall/CRC

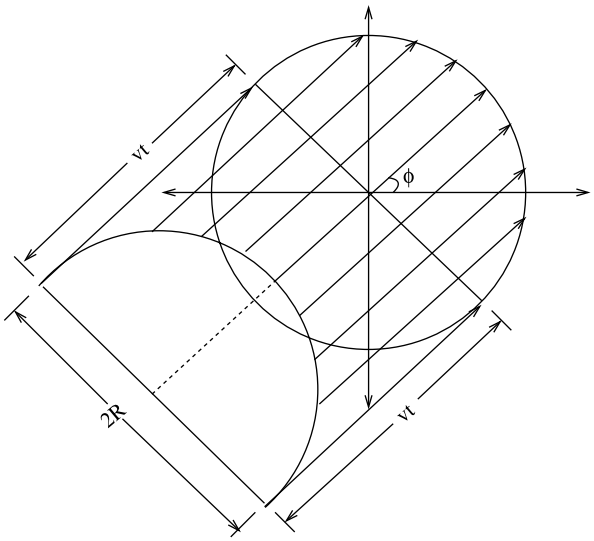


Figure 50.10. Calculation of link breakage interarrival time distribution.

in the figure consists of all points not more than vt meters away along angle ϕ from the semicircle $\alpha \in [\frac{\pi}{2} - \phi, \frac{3\pi}{2} - \phi]$. It is easy to see that a node moving at an angle ϕ can break a link with node 1 only by moving out of its transmission zone from a point on this semicircle. Given its relative velocity $\vec{v} = v\cos\phi\hat{i} + v\sin\phi\hat{j}$, a node will leave the transmission zone of node 1 within the next t seconds — and thus breaking the link between the two — if it is currently located in $\mathbf{D_b}$. Note that $\mathbf{D_b}$ also includes the possibility of nodes that are currently outside the transmission zone of node 1 and have yet to form a link with it.

The area of the shaded region $\mathbf{D_b}$ is $A = 2Rvt$. For given v and ϕ, the probability that the link breakage interarrival time is not more than t is equal to the probability that there is at least one node in $\mathbf{D_b}$ with velocity $\vec{v}$.

$$\begin{aligned}\text{Prob}\{\text{link breakage interarrival time} \le t|v, \phi\} &= \text{Prob}\{\text{at least one node in } \mathbf{D_b}|\mathbf{v}, \phi\} \\ &= 1 - \mathrm{e}^{-\sigma 2Rvt}\end{aligned} \tag{50.33}$$

Thus, the CDF of link breakage interarrival time is given by

$$\begin{aligned}F^{v_1}_{\text{break}}(t) &= \text{Prob }\{\text{link breakage interarrival time} \le t\} \\ &= \iint\limits_{v,\phi} \left(1 - \mathrm{e}^{-2R\sigma tv}\right) f(v,\phi)\,\mathrm{d}v\,\mathrm{d}\phi\end{aligned} \tag{50.34}$$

The right-hand sides of Equations (50.27) and (50.34) are the same, implying that the distributions of link breakage interarrival time and new link interarrival time are the same:

$$\begin{aligned}F^{v_1}_{\text{break}}(t) &= F^{v_1}_{\text{arrival}}(t) \\ &= 1 - \frac{1}{\pi(b-a)} \int_0^{\pi} \int_0^{\infty} \mathrm{e}^{-2R\sigma tv} \frac{v}{\sqrt{v^2 + v_1^2 + 2vv_1\cos\phi}} \\ &\quad \left[u\left(\sqrt{v^2 + v_1^2 + 2vv_1\cos\phi} - a\right) - u\left(\sqrt{v^2 + v_1^2 + 2vv_1\cos\phi} - b\right)\right] \mathrm{d}v\,\mathrm{d}\phi\end{aligned} \tag{50.35}$$

© 2005 by Chapman & Hall/CRC

Note that, using a different argument, it was already shown in Section 50.3.5 that the expected rate of link breakages is equal to the expected rate of new link arrivals.

50.3.7 Link Change Interarrival Time Distribution

Creation of a new link or expiry of an old link constitutes a change in the set of links of a node. Given its relative velocity $\vec{v} = v\cos\phi\hat{i} + v\sin\phi\hat{j}$, the existence of a node in the shaded region $\mathbf{D_a}$ of Figure 50.5 will cause the formation of a new link within the next t seconds. Likewise, a node with velocity $\vec{v}$ in the shaded region $\mathbf{D_b}$ of Figure 50.10 will cause the breaking of a link within the next t seconds. Figure 50.11 shows the union of these two shaded regions, $\mathbf{D_c} = \mathbf{D_a} \cup \mathbf{D_b}$. Given $\vec{v}$, a node currently located in the shaded region $\mathbf{D_c}$ of Figure 50.11 will cause a link change within the next t seconds.

The area A of $\mathbf{D_c}$ can be expressed as

$$A = \begin{cases} 2vtR + 2R^2\left(\sin^{-1}\left(\frac{vt}{2R}\right) + \frac{vt}{2R}\sqrt{1-\left(\frac{vt}{2R}\right)^2}\right) & \text{if } vt \le 2R \\ 2vtR + \pi R^2 & \text{if } vt > 2R \end{cases} \tag{50.36}$$

From Assumption 5, as the nodes are assumed to be Poisson distributed, we have

$$\text{Prob}\{\text{no node in } \mathbf{D_c} | v, \phi\} = \mathrm{e}^{-\sigma A} \tag{50.37}$$

Therefore, the link change interarrival time distribution is given by

$$\begin{aligned} F^{v_1}_{\text{change}}(t) &= \text{Prob}\{\text{link change interarrival time} \le t\} \\ &= 1 - \text{Prob}\{\text{link change interarrival time} > t\} \\ &= 1 - \text{Prob}\{\text{new link interarrival time} > t, \\ &\qquad \text{link breakage interarrival time} > t\} \\ &= 1 - \iint\limits_{v,\phi} \text{Prob}\{\text{no node in } \mathbf{D_c} | v, \phi\} f(v, \phi)\, \mathrm{d}v\, \mathrm{d}\phi \\ &= 1 - \iint\limits_{v,\phi} \mathrm{e}^{-\sigma A} f(v, \phi) \mathrm{d}v\, \mathrm{d}\phi \\ &= 1 - \frac{1}{\pi(b-a)} \left\{ \int_0^{\pi} \int_{2R/t}^{\infty} \mathrm{e}^{-\sigma(2vtR+\pi R^2)} \frac{v}{\sqrt{v^2 + v_1^2 + 2vv_1\cos\phi}} \right. \\ &\qquad \left[u\left(\sqrt{v^2 + v_1^2 + 2vv_1\cos\phi} - a\right) - u\left(\sqrt{v^2 + v_1^2 + 2vv_1\cos\phi} - b\right) \right] \mathrm{d}v\, \mathrm{d}\phi \\ &\qquad + \int_0^{\pi} \int_0^{2R/t} \mathrm{e}^{-\sigma\left\{2vtR + 2R^2\left[\sin^{-1}(vt/2R) + vt/2R\sqrt{1-(vt/2R)^2}\right]\right\}} \frac{v}{\sqrt{v^2 + v_1^2 + 2vv_1\cos\phi}} \\ &\qquad \left. \left[u\left(\sqrt{v^2 + v_1^2 + 2vv_1\cos\phi} - a\right) - u\left(\sqrt{v^2 + v_1^2 + 2vv_1\cos\phi} - b\right) \right] \mathrm{d}v\, \mathrm{d}\phi \right\} \end{aligned} \tag{50.38}$$

It is not possible to evaluate the integrals in Equation (50.38) explicitly. In Figure 50.12 we plot the link change interarrival time distribution $F^{v_1}_{\text{change}}(t)$ for different node velocities v_1. $a = 0$ m/s, $b = 40$ m/s, $R = 250$ m and $\sigma = 10/\pi R^2$ nodes/m^2 have been used for the figure. In Figure 50.13, the corresponding link change interarrival time probability density $f^{v_1}_{\text{change}}(t)$ is plotted for the same parameter values.

© 2005 by Chapman & Hall/CRC

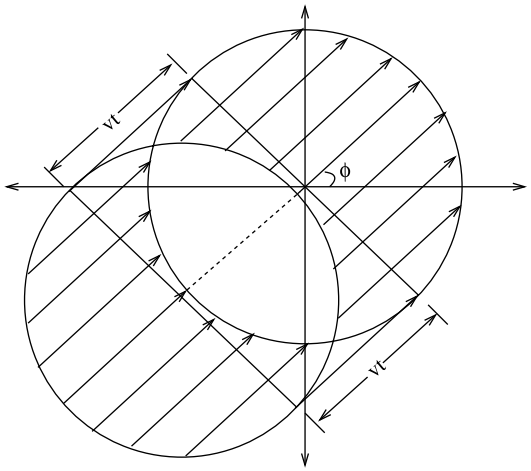


Figure 50.11. Calculation of link change interarrival time distribution.

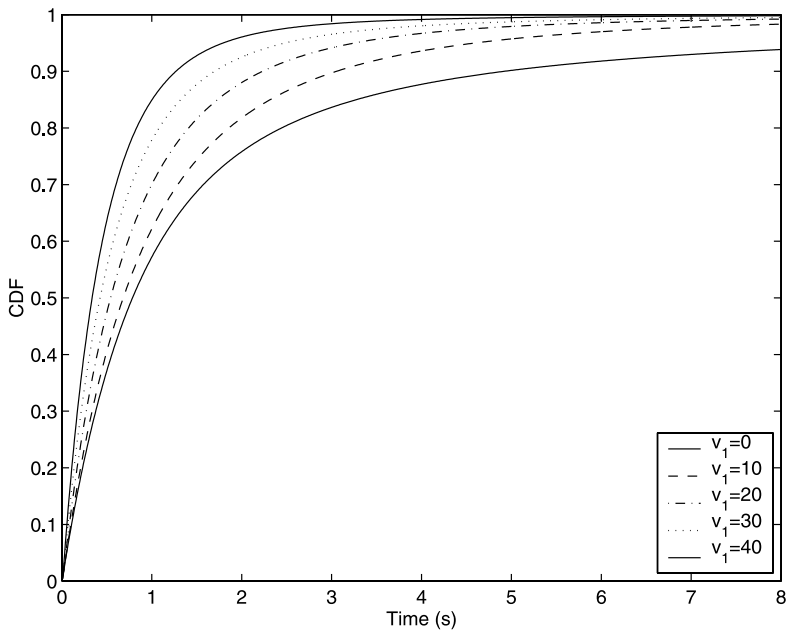


Figure 50.12. The CDF of link change interarrival time for a node moving with velocity v_1, for $a = 0$ m/s, $b = 40$ m/s, $R = 250$ m and $\sigma = 10/\pi R^2$.

It can be readily observed from the figure that the link change interarrival time density function decreases rapidly as time t increases. It is interesting to compare Figure 50.8 and Figure 50.13, which plot the PDFs of new link interarrival time (or link breakage interarrival time) and link change interarrival time respectively. The curves in Figure 50.13 appear to be scaled versions (by a factor of approximately 2, and then normalized) of the curves in Figure 50.8.

50.3.8 Expected Number of Neighbors

As the locations of nodes in the network are modeled as Poisson distributed random variables with intensity σ, the expected number of nodes located in an area A is equal to σA. This implies that the

© 2005 by Chapman & Hall/CRC

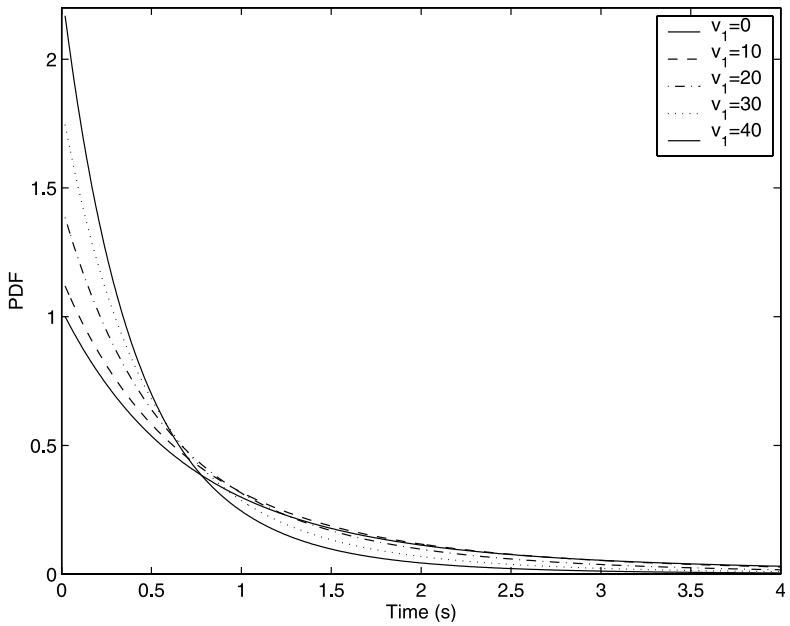


Figure 50.13. The PDF of link change interarrival time for a node moving with velocity v_1, for $a = 0$ m/s, $b = 40$ m/s, $R = 250$ m and $\sigma = 10/\pi R^2$.

expected number of nodes in the transmission zone of a particular node is equal to $\sigma\pi R^2$. Therefore, the expected number of neighbors of a node is given by

$$\bar{N} = \sigma\pi R^2 - 1 \tag{50.39}$$

As expected, $\bar{N}$ increases with node density σ, but is independent of node mobility.

50.4 Simulations

In this section we illustrate the validity of the analytically derived expressions of the link properties by comparing them with corresponding statistics collected from simulations.

The simulation set-up is as follows. The network consists of 200 nodes, where each node has a transmission radius R of 250 m. These nodes are initially spread randomly over a square region, whose sides are chosen to be equal to 1981.7 m each so that the node density σ turns out to be equal to $10/\pi R^2$ nodes/m^2 (or equivalently, $\rho = 10$). The velocity of the nodes is chosen to be uniformly distributed between $a = 0$ m/s and $b = 40$ m/s. A node's velocity is initially assigned a direction θ, which is uniformly distributed between 0 and 2π. When a node reaches an edge of the square simulation region it is reflected back into the network area by setting its direction to $-\theta$ (horizontal edges) or $\pi - \theta$ (vertical edges). The magnitude of its velocity is not altered. The simulation duration is set to 240 min.

Statistics characterizing the link properties as a function of the node velocity v_1 are collected from the simulations. For the plots of Figures 50.14(ii), 50.15(ii) and 50.16(ii), the heights of the frequency bars have been normalized to make the total area covered by each of the histogram plots equal to unity.

From Figures 50.14, 50.15 and 50.16, one can see that the theoretical curves are in fairly close agreement with the simulation results. The difference between the two is mainly attributed to the boundary effect present in the simulations. Nodes close to the boundary of the square simulation region experience fewer (or possibly, no) node arrivals from the direction of the boundary than otherwise expected. Also, when a node reaches the boundary of the simulation region it gets reflected back into the network. Additional simulation studies suggest that the gap between the analytical and the experimental results decreases as the network size is increased while keeping the node density constant.

© 2005 by Chapman & Hall/CRC

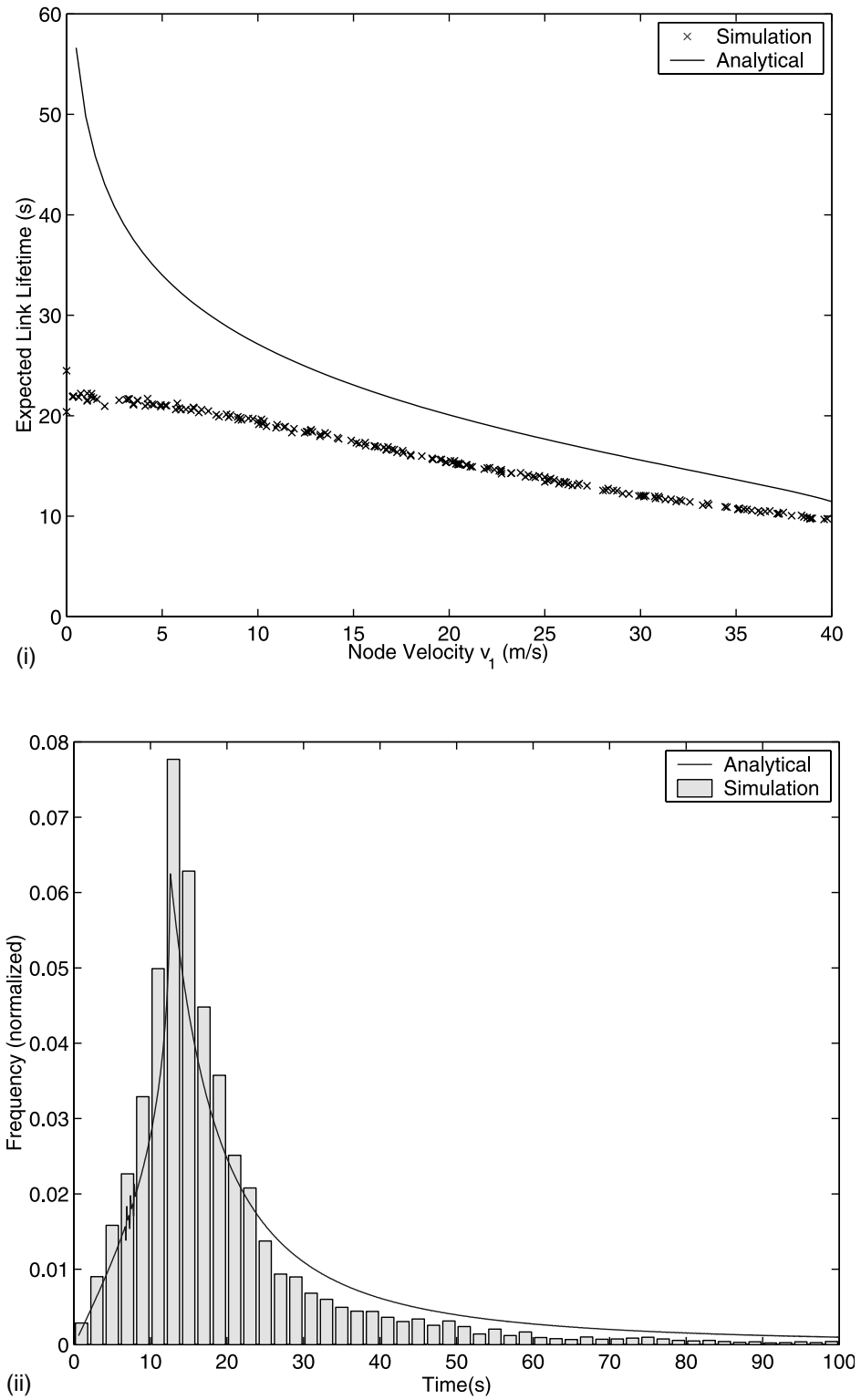
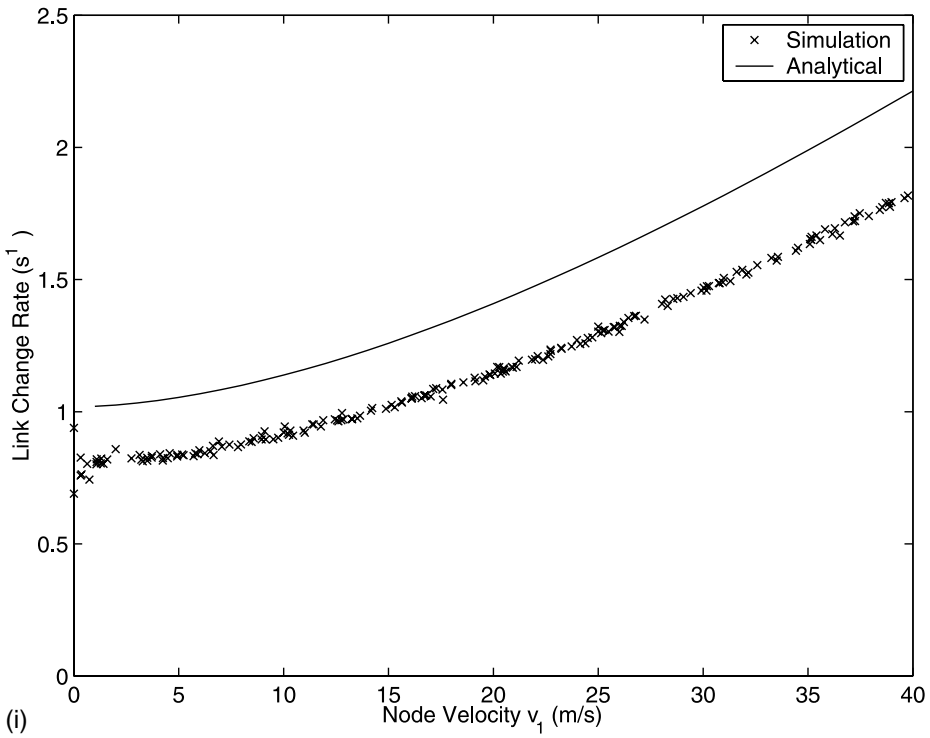


Figure 50.14. Comparison with simulation statistics: (i) expected link lifetime; (ii) link lifetime PDF for a node with velocity $v_1 = 0\,\mathrm{m/s}$.

© 2005 by Chapman & Hall/CRC



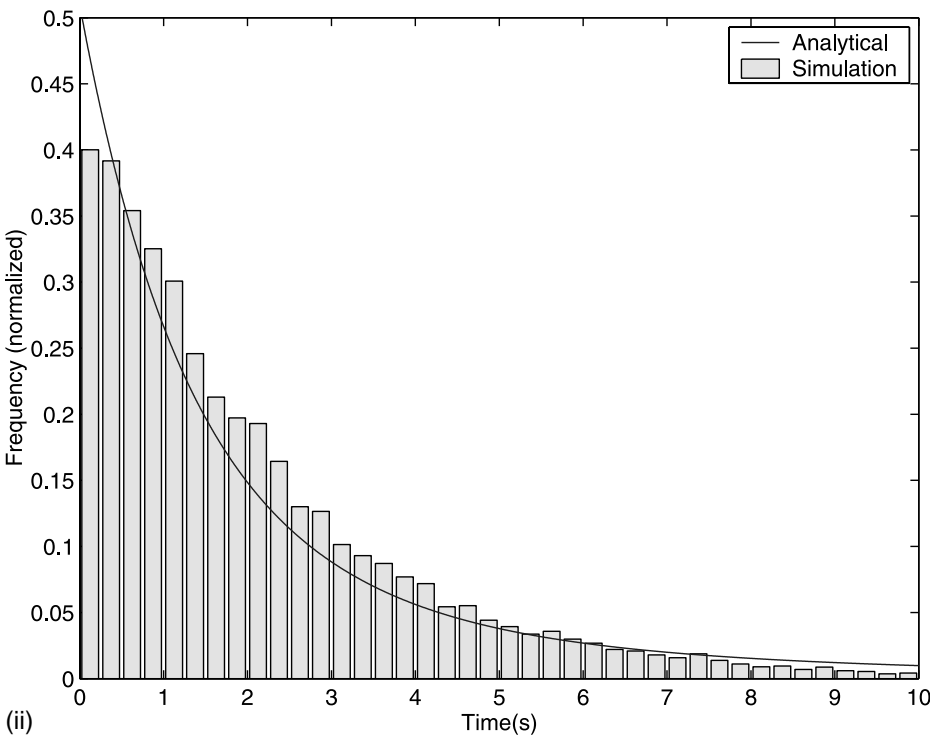


Figure 50.15. Comparison with simulation statistics: (i) expected new link arrival rate; (ii) new link interarrival time PDF for a node with velocity $v_1 = 0$ m / s.

© 2005 by Chapman & Hall/CRC

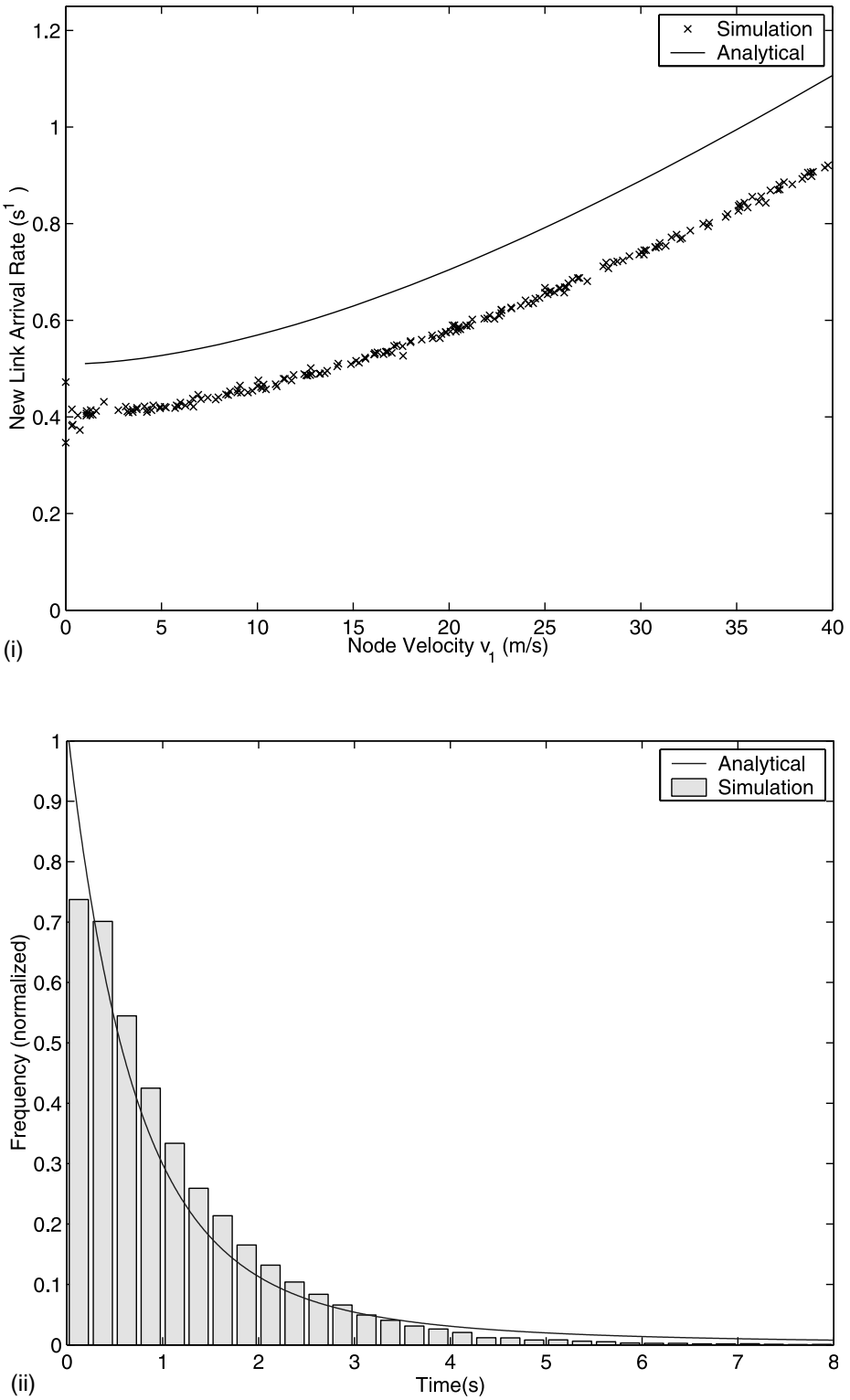


Figure 50.16. Comparison with simulation statistics: (i) expected link change rate; (ii) link change interarrival time PDF for a node with velocity $v_1 = 0$ m / s.

© 2005 by Chapman & Hall/CRC

The results for the expected link breakage rate and the link breakage interarrival time density are similar to those in Figure 50.15 and are, therefore, omitted here.

50.5 Applications of Link Properties

The various properties investigated in Section 50.3 characterize the behavior of the links of a node in a mobile environment. The derived properties can be used to design efficient algorithms for communication in *ad hoc* and sensor networks. They can also be used as a basis for analyzing the performance bounds of network protocols. In this section, we discuss some representative applications of the link properties studied in the previous section.

The link lifetime distribution can be used to examine the stability of links in the network. Once communication starts over a link, its residual lifetime distribution can be calculated as a function of the link lifetime distribution. Mathematically, the probability density $r_T^{v_1}(t)$ of residual link lifetime given that the link has been in existence for T seconds already can be expressed as

$$r_T^{v_1}(t) = \frac{f_{\text{link}}^{v_1}(t+T)}{1 - F_{\text{link}}^{v_1}(T)} \tag{50.40}$$

Here, $f_{\text{link}}^{v_1}(\cdot)$ and $F_{\text{link}}^{v_1}(\cdot)$ are the link lifetime PDF and CDF respectively, as derived in Section 50.3.2.

The residual link lifetime density can be used to evaluate the lifetime of a route in the network. For example, consider a route with K links and let $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_K$ be the random variables representing each of their residual lifetimes at the time when the route is formed, given that the links have already been in existence for $T_1, T_2, \ldots, T_K$ seconds respectively. Let $\mathbf{Y}$ be a random variable representing the lifetime of the route formed by the K links. As the route is deemed to have failed when any of the K links breaks, the route lifetime can be expressed as the minimum of the lifetimes of its constituent links:

$$\mathbf{Y} = \min(\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_K) \tag{50.41}$$

If we assume that the residual link lifetimes are independent and identically distributed, then the distribution $F_Y(t)$ of $\mathbf{Y}$ can be calculated as

$$\begin{aligned} F_Y(t) &= \text{Prob}\{\mathbf{Y} \le t\} \\ &= 1 - \text{Prob}\{\min(\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_K) > t\} \\ &= 1 - \text{Prob}\{\mathbf{X}_1 > t, \mathbf{X}_2 > t, \ldots, \mathbf{X}_K > t\} \\ &= 1 - \text{Prob}\{\mathbf{X}_1 > t\}\,\text{Prob}\{\mathbf{X}_2 > t\} \cdots \text{Prob}\{\mathbf{X}_K > t\} \\ &= 1 - \left(1 - R_{T_1}^{v_{1_1}}(t)\right)\left(1 - R_{T_2}^{v_{1_2}}(t)\right) \cdots \left(1 - R_{T_K}^{v_{1_K}}(t)\right) \end{aligned} \tag{50.42}$$

where $R_{T_i}^{v_{1_i}}(t)$ is the cumulative distribution function of the residual link lifetime of the ith link in the route, whose upstream node is moving with velocity v_{1_i}, given that the link was formed T_i seconds ago. $R_{T_i}^{v_{1_i}}(t)$ can be evaluated by integrating the corresponding density in Equation (50.40).

The route lifetime distribution can be used to analyze the performance of routing protocols in *ad hoc* and sensor networks. It can also be used to provide quality of service (QoS) in the network. For example, the above framework can form the basis of schemes for selection of the best[3] set of routes for QoS techniques like multi-path routing [13] and alternate path routing [14].

[3]In terms of the particular QoS metric under consideration.

© 2005 by Chapman & Hall/CRC

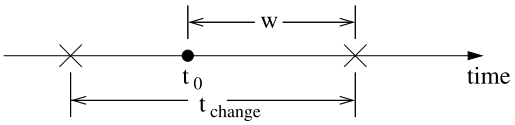


Figure 50.17. Timeline where the "×"s represent the arrival of link changes and t_0 is a fixed point.

Another application of the link properties is the optimal selection of the time-to-live interval of route caches in on-demand routing protocols. For example, the work in Liang and Haas [15] can be supplemented using the distributions derived in this chapter to minimize the expected routing delay. It is also possible to develop alternate schemes to optimize other network performance metrics, if so desired.

Renewal theory [16] can be used to characterize the residual time $\mathbf{w}$ to arrival of the next link change after a given fixed instant t_0. Figure 50.17 shows the timeline where t_0 and $\mathbf{w}$ are indicated and the "×"s represent the arrival of link changes. The probability density $f_w^{v_1}(w)$ of $\mathbf{w}$ is given by

$$f_w^{v_1}(w) = \dot{\gamma}(v_1)\left[1 - F_{\text{change}}^{v_1}(w)\right] \tag{50.43}$$

where $\dot{\gamma}(v_1)$ and $F_{\text{change}}^{v_1}(w)$ are the expected link change arrival rate and the link change interarrival time distribution respectively, as found before. Similarly, given a fixed point t_0, the density of the residual time to arrival of the next new link or the next link breakage can be calculated by appropriately replacing the corresponding functions in Equation (50.43).

Strategies for broadcasting of routing updates by proactive routing protocols have been proposed by Samar and Haas [17]. These updating strategies are shown to lead to a considerable reduction in routing overhead while maintaining good performance in terms of other metrics. The design of these updating strategies is based on the assumption that link change interarrival times are exponentially distributed. However, the actual link change interarrival time distribution experienced by the nodes has been derived in Section 50.3.7. These updating strategies can be redesigned by utilizing the more realistic distributions as derived here. This would further improve the performance offered by these updating strategies.

50.6 Conclusions

Developing efficient algorithms for communication in multi-hop environments like *ad hoc* and sensor networks is challenging, particularly due to the mobility of nodes forming the network. An attempt has been made in this chapter to develop an analytical framework that can provide a better understanding of network behavior under mobility. We derive expressions for a number of properties characterizing the creation, lifetime, and expiration of communication links in the network.

Not only can this study help analyze the performance of network protocols, it can also assist in developing efficient schemes for communication. This has been illustrated by the discussion on a few example applications of the derived link properties.

Appendix 50A

50A.1 Joint Probability Density of v, ϕ and α

Here, we derive the joint PDF $f_{v\phi\alpha}(v, \phi, \alpha)$ for the nodes that enter the transmission zone of node 1:

$$f_{v\phi\alpha}(v, \phi, \alpha) = f_{\alpha|v\phi}(\alpha|v, \phi) f_{v\phi}(v, \phi) \tag{50A.1}$$

© 2005 by Chapman & Hall/CRC

$f_{\alpha|v\phi}(\alpha|v, \phi)$ is the conditional PDF of the angle α defining node 2's point of entry $(-R\cos\alpha, R\sin\alpha)$ into the transmission zone of node 1, given its relative velocity $\vec{v} = v\cos\phi\hat{i} + v\sin\phi\hat{j}$.[4] Now, given the direction ϕ of node 2's relative velocity, the node can only enter the zone from a point on the semicircle $\alpha \in [-(\frac{\pi}{2} + \phi), \frac{\pi}{2} - \phi]$. Consider the diameter of this semicircle, which is perpendicular to the direction of node 2's relative velocity. As nodes in the network are assumed to be randomly distributed, a node entering the zone with velocity $\vec{v}$ can intersect this diameter at any point on it with equal probability. This is illustrated in Figure 50A.1, where the node's trajectory is equally likely to intersect the diameter QR at any point Q, $P1$, $P2$, ..., R on it, indicating that the probability of location of this point of intersection is uniformly distributed on the diameter.

In Figure 50A.2, node 2 enters the transmission zone at T and travels along TV, which makes an angle ϕ with the horizontal. OT makes an angle α with OX''. QR is the diameter perpendicular to TV, defining

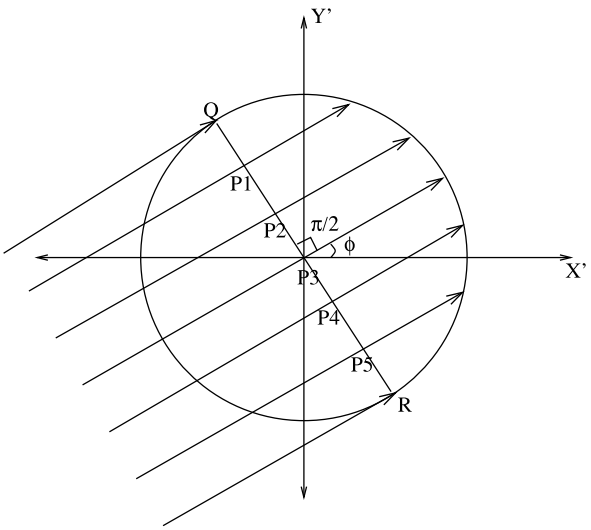


Figure 50A.1. Given the direction of a node's relative velocity, it can intersect the diameter QR at any point on it with equal probability.

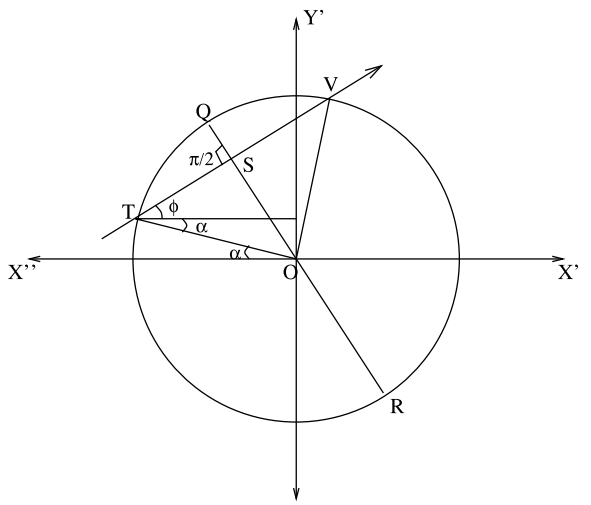


Figure 50A.2. Calculation of $f_{\alpha|v\phi}(\alpha|v, \phi)$.

[4]Note that α is measured clockwise from the negative X axis.

© 2005 by Chapman & Hall/CRC

the semicircle $\alpha \in [-(\frac{\pi}{2}+\phi),\ \frac{\pi}{2}-\phi]$. Let $OS = r$, where S is the point of intersection of TV and QR. As $OT = OV = R$, it is easy to see that $r = R\sin(\alpha+\phi)$.

Let α be the random variable representing the angle defining the point of entry of node 2 in the zone. For $\alpha \in [-(\frac{\pi}{2}+\phi),\ \frac{\pi}{2}-\phi]$:

$$\begin{aligned} F_{\alpha|v\phi}(\alpha|v,\phi) &= \text{Probability}\,\{\alpha \le \alpha|v,\ \phi\} \\ &= \int_{-R}^{r} \frac{1}{2R}\mathrm{d}r \\ &= \frac{r+R}{R} \\ &= \frac{1}{2}[1+\sin(\alpha+\phi)] \end{aligned} \tag{50A.2}$$

Hence, by differentiating Equation (50A.2):

$$\begin{aligned} f_{\alpha|v\phi}(\alpha|v,\phi) &= \begin{cases} \frac{1}{2}\cos(\alpha+\phi) & \alpha \in \left[-(\frac{\pi}{2}+\phi),\ \frac{\pi}{2}-\phi\right] \\ 0 & \text{otherwise} \end{cases} \\ &= \frac{1}{2}\cos(\alpha+\phi)\left\{u\left[\alpha+\left(\frac{\pi}{2}+\phi\right)\right]-u\left[\alpha-\left(\frac{\pi}{2}-\phi\right)\right]\right\} \end{aligned} \tag{50A.3}$$

where $u(\cdot)$ is the unit step function.

Note that for $\alpha \in [-(\frac{\pi}{2}+\phi),\ \frac{\pi}{2}-\phi]$, $\cos(\alpha+\phi) \ge 0\ \forall\ \phi \in [-\pi,\ \pi]$.

$f_{v\phi}(v,\phi)$ is the joint PDF of v and ϕ for the nodes that enter the zone. This is simply the density of the relative velocity $\vec{v}$ of the nodes in the network. It can be calculated by

$$f_{v\phi}(v,\phi) = \frac{f_{v_2\theta}(v_2^*,\theta^*)}{|J(v_2^*,\theta^*)|} \tag{50A.4}$$

where $f_{v_2\theta}(v_2^*,\theta^*)$ is the joint PDF of v_2 and θ, v_2^* and θ^* are the values of v_2 and θ that satisfy Equations (50.3) and (50.4), and

$$J(v_2,\theta) = \begin{vmatrix} \dfrac{\partial v}{\partial v_2} & \dfrac{\partial v}{\partial \theta} \\ \dfrac{\partial \phi}{\partial v_2} & \dfrac{\partial \phi}{\partial \theta} \end{vmatrix} \tag{50A.5}$$

is the *Jacobian* for the transformation.

Solving Equations (50.3) and (50.4) for v_2^* and θ^* gives

$$\theta^* = \tan^{-1}\left(\frac{\sin\phi}{\cos\phi + v_1/v}\right) \tag{50A.6}$$

$$v_2^* = \sqrt{v^2 + v_1^2 + 2vv_1\cos\phi} \tag{50A.7}$$

© 2005 by Chapman & Hall/CRC

Using Equations (50.3) and (50.4) to get the derivatives for the *jacobian*,

$$\begin{aligned} J(v_2, \theta) &= \begin{vmatrix} \dfrac{v_2 - v_1\cos\theta}{\sqrt{v_1^2 + v_2^2 - 2v_1v_2\cos\theta}} & \dfrac{v_1v_2\sin\theta}{\sqrt{v_1^2 + v_2^2 - 2v_1v_2\cos\theta}} \\ \dfrac{-v_1\sin\theta}{v_1^2 + v_2^2 - 2v_1v_2\cos\theta} & \dfrac{v_2^2 - v_1v_2\cos\theta}{v_1^2 + v_2^2 - 2v_1v_2\cos\theta} \end{vmatrix} \\ &= \frac{v_2}{\sqrt{v_1^2 + v_2^2 - 2v_1v_2\cos\theta}} \end{aligned} \tag{50A.8}$$

Therefore:

$$J(v_2^*, \theta^*) = \frac{\sqrt{v^2 + v_1^2 + 2vv_1\cos\phi}}{v} \tag{50A.9}$$

From Assumption 2, v_2 is uniformly distributed between a and b. Also, from Assumption 3, θ is uniformly distributed between 0 and 2π. Thus, their individual PDF are given by

$$f_{v_2}(v_2) = \frac{1}{b-a}\Big[u(v_2 - a) - u(v_2 - b)\Big] \tag{50A.10}$$

$$f_\theta(\theta) = \frac{1}{2\pi} \tag{50A.11}$$

As v_2 and θ are assumed to be independent (Assumption 4), their joint PDF is simply the product of their individual density functions:

$$f_{v_2\theta}(v_2^*, \theta^*) = \frac{1}{2\pi(b-a)}\Big[u(v_2^* - a) - u(v_2^* - b)\Big] \tag{50A.12}$$

Therefore, using Equation (50A.4), we get

$$\begin{aligned} f_{v\phi}(v, \phi) = &\frac{1}{2\pi(b-a)}\frac{v}{\sqrt{v^2 + v_1^2 + 2vv_1\cos\phi}} \\ &\left[u\left(\sqrt{v^2 + v_1^2 + 2vv_1\cos\phi} - a\right) - u\left(\sqrt{v^2 + v_1^2 + 2vv_1\cos\phi} - b\right)\right] \end{aligned} \tag{50A.13}$$

Hence, from Equations (50A.1), 50A.3) and (50A.13):

$$\begin{aligned} f_{v\phi\alpha}(v, \phi, \alpha) &= f_{\alpha|v\phi}(\alpha|v, \phi) f_{v\phi}(v, \phi) \\ &= \left\{ \frac{1}{2\pi(b-a)}\frac{v}{\sqrt{v^2 + v_1^2 + 2vv_1\cos\phi}}\left[u\left(\sqrt{v^2 + v_1^2 + 2vv_1\cos\phi} - a\right)\right.\right. \\ &\quad \left.\left. - u\left(\sqrt{v^2 + v_1^2 + 2vv_1\cos\phi} - b\right)\right]\right\}\left(\frac{1}{2}\cos(\alpha + \phi)\left\{u\left[\alpha + \left(\frac{\pi}{2} + \phi\right)\right]\right.\right. \\ &\quad \left.\left. - u\left[\alpha - \left(\frac{\pi}{2} - \phi\right)\right]\right\}\right) \end{aligned} \tag{50A.14}$$

Finally, the joint density of v, ϕ and α is given by Equation (50A.14).

© 2005 by Chapman & Hall/CRC

References

[1] Goldsmith, A.J. and Wicker, S.B., Design challenges for energy-constrained *ad hoc* wireless networks, *IEEE Wireless Communications*, 9(4), 8, 2002.

[2] Toh, C.-K., Associativity-based routing for *ad hoc* networks, *Wireless Personal Communications*, March, 4(2), 103, 1997.

[3] Dube, R. *et al.*, Signal stability based adaptive routing (SSA) for *ad hoc* networks, *IEEE Personal Communications*, 4(1), 36, 1997.

[4] Agarwal, S. *et al.*, Route-lifetime assessment based routing (RABR) protocol for mobile *ad-hoc* networks, in *IEEE International Conference on Communications 2000*, vol. 3, New Orleans, 2000, 1697.

[5] Su, W. *et al.*, Mobility prediction and routing in *ad hoc* wireless networks, *International Journal of Network Management*, 11(1), 3, 2001.

[6] Gerharz, M. *et al.*, Link stability in mobile wireless *ad hoc* networks, in *IEEE Conference on Local Computer Networks (LCN'02)*, Tampa, FL, November 2002.

[7] Lim, G. *et al.*, Link stability and route lifetime in *ad-hoc* wireless networks, in *2002 International Conference on Parallel Processing Workshops (ICPPW'02)*, Vancouver, Canada, August 2002.

[8] Hu, Y.-C. and Johnson, D.B., Caching strategies in on-demand routing protocols for wireless *ad hoc* networks, in *IEEE/ACM International Conference on Mobile Computing and Networking (MobiCom 2000)*, Boston, MA, August 6–11, 2000.

[9] Camp, T. *et al.*, A survey of mobility models for *ad hoc* network research, *Wireless Communication & Mobile Computing (WCMC): Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications*, 2(5), 483, 2002.

[10] Turgut, D. *et al.*, Longevity of routes in mobile *ad hoc* networks, in *VTC Spring 2001*, Rhodes, Greece, May 6–9, 2001.

[11] McDonald, A.B. and Znati, T.F., A mobility-based framework for adaptive clustering in wireless *ad hoc* networks, *IEEE Journal of Selected Areas in Communications*, 17(8), 1466, 1999.

[12] Jiang, S. *et al.*, A prediction-based link availability estimation for mobile *ad hoc* networks, in *IEEE INFOCOM 2001*, Anchorage, AK, April 22–26, 2001.

[13] Papadimitratos, P. *et al.*, Path set selection in mobile *ad hoc* networks, in *ACM Mobihoc 2002*, Lausanne, Switzerland, June 9–11, 2002.

[14] Pearlman, M.R. *et al.*, On the impact of alternate path routing for load balancing in mobile *ad hoc* networks, in *ACM MobiHOC'2000*, Boston, MA, August 11, 2000.

[15] Liang, B. and Haas, Z.J., Optimizing route-cache lifetime in *ad hoc* networks, in *IEEE INFOCOM 2003*, San Francisco, CA, March 30–April 3, 2003.

[16] Papoulis, A., *Probability, Random Variables, and Stochastic Processes*, 3rd ed., McGraw-Hill, 1991.

[17] Samar, P. and Haas, Z.J., Strategies for broadcasting updates by proactive routing protocols in mobile *ad hoc* networks, in *IEEE MILCOM 2002*, Anaheim, CA, October 2002.

© 2005 by Chapman & Hall/CRC

VIII

System Control

51. Example Distributed Sensor Network Control Hierarchy
Mengxia Zhu, S.S. Iyengar, Jacob Lamb, R.R. Brooks, and Matthew Pirretti 977
Introduction • Petri Nets • Hierarchy Models • Control Specifications • Controller Design • Case Study • Discussion and Conclusions • Acknowledgments and Disclaimer • Appendix

Wireless sensor networks (WSNs) are an important military technology with civil and scientific applications. This chapter emphasizes on deriving models and controllers for distributed sensor networks (DSNs) consisting of multiple cooperating nodes and each battery powered node has wireless communications, local processing capabilities, sensor inputs, data storage and limited mobility.

Zhu *et al.* focused on deriving a discrete event controller system for distributed surveillance networks that consists of three interacting hierarchies — sensing, communications, and command. The focus of their work is on deriving controllers using the methods:

(i) Petri Net
(ii) Finite state machine (FSM) using the Ramadge and Wonham approach
(iii) Vector addition control using the Wonham and Li approach

They compare the controllers in terms of expressiveness and performance and showed that Petri Net model is concise and efficient, whereas the FSM model requires an offline state search, but its online implementation is less complex and the vector addition controller is essentially a Petri Net controller that enforces inequality constraints upon the system at runtime. They also present innovation for deriving FSM controller which benefits from the use of Karp-Miller tree to represent all possible evolutions of the Petri Net plant model from an initial marking, and Moore machine to generate control patterns in terms of current encoded state automatically.

In summary, this section elaborates on deriving a discrete event controller system for distributed surveillance networks.

© 2005 by Chapman & Hall/CRC

51

Example Distributed Sensor Network Control Hierarchy

Mengxia Zhu, S.S. Iyengar, Jacob Lamb, R.R. Brooks, and Matthew Pirretti

51.1 Introduction

In this chapter we derive models of, and controllers for, distributed sensor networks (DSNs) consisting of multiple cooperating nodes. Each battery-powered node has wireless communications, local processing capabilities, sensor inputs, data storage, and limited mobility. An individual node would be capable of isolated operation, but practical deployment scenarios require coordination among multiple nodes. We are particularly interested in self-organization technologies for these systems. Network self-configuration is needed for the system to adapt to a changing environment [1]. In this chapter we derive hierarchical structures that support user control of the distributed system.

Our model uses discrete event dynamic systems (DEDS) formalisms. DEDS have discrete time and state spaces. They are usually asynchronous and nondeterministic. Many DEDS modeling and control methodologies exist and no dominant paradigm has emerged [2]. We use Petri nets, as will be described in Section 51.2, to model the plants to be controlled. Our sensor network model has three intertwined hierarchies, which evolve independently. We derive controllers to enforce system consistency constraints across the three hierarchies. Three equivalent controllers are derived using (i) Petri net, (ii) vector addition and (iii) finite-state machine (FSM) techniques. We compare the controllers in terms of expressiveness and performance. Innovative use of Karp–Miller trees [3] allows us to derive FSM controllers for the Petri net plant model. In addition, we show how FSM controllers can be derived automatically from control specifications in the proper format.

The remainder of the chapter is organized as follows. Section 51.2 gives a review of Petri nets. Section 51.3 describes the structure of the network hierarchies. In Section 51.4 we provide control specifications. The controllers are derived in Section 51.5. Section 51.5 also provides brief tutorials on each approach. Section 51.6 provides experimental results from simulations run using the controllers.

51.2 Petri Nets

Carl Adam Petri defined Petri nets as a graphic mathematical model for describing information flow in 1962. This model proved versatile in visualizing and analyzing the behavior of asynchronous,

© 2005 by Chapman & Hall/CRC

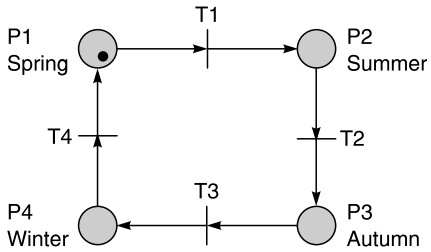


Figure 51.1. Petri net model of the cycle of the seasons with four possible markings: {1000, 0100, 0010, 0001}.

concurrent systems. Later research led to the direct application of Petri nets in automata theory. Petri nets are excellent for modeling the relationship between events, resources, and system states [4].

A Petri net is a bi-partite graph with two classes of nodes: *places* and *transitions*. The number of places and transitions are finite and nonzero. Directed arcs connect nodes. Arcs either connect a transition to a place or a place to a transition. Arcs can have an associated integer weight. DEDS state variables are represented by places. Events are represented by transitions. Places contain *tokens.*

The DEDS state space is defined by the *marking* of the Petri net. A marking is a vector expressing the number of tokens in each place. A transition is enabled when the places with arcs incident to the transition all contain at least as many tokens as the weight of the associated arcs. The firing of a transition removes tokens from all places with arcs incident to the transition and deposits tokens in all places with arcs issuing from the transition. The number of tokens removed (added) is equal to the weight of the associated arc. The firing of a transition thus changes the marking of the Petri net and the state of the DEDS. Transitions fire one at a time even when more than one transition has been enabled. The system is nondeterministic, in that any enabled transition can fire.

Mathematically, a Petri net is represented as the tuple $S = (P, T, I, O, u)$ with P the finite set of places, T the finite set of transitions, I the finite set of arcs from places to transitions, O the finite set of arcs from transitions to places, and u is an integer vector representing the current marking [3]. Figure 51.1 is a simple example of a Petri net modeling the cycle of seasons. Safeness is a special issue to be considered. A Petri net is safe if all places contain no more than one token. However, a Petri net is called k-safe or k-bounded if no place contains no more than k tokens. An unbounded Petri net may contain an infinite number of tokens in places and may have an infinite number of markings. Conversely, a bounded Petri net is essentially an FSM with each node corresponding to every reachable state. In deriving our controllers, we derive Karp–Miller trees from the Petri nets [3]. Despite their name, Karp–Miller trees are graph structures; they represent all possible markings a Petri net can reach from a given initial marking and ω is usually used to represent an infinite number of tokens in a place if necessary. The algorithm for deriving the Karp–Miller tree is given in Section 51.5.

51.3 Hierarchy Models

51.3.1 Overview and Terminology

In an effort to describe thoroughly the functionality of a remote, multi-modal, mobile sensing network, three issues must be addressed:

- Network communication — maintaining communications within the network.
- Collaborative sensing — coordinating sensor data interpretation.
- Operational command — assigning resources within the network and controlling internal system logistics.

© 2005 by Chapman & Hall/CRC

Each hierarchy is composed of three separate levels:

- *Root*. This is the top level of the hierarchy. It coordinates among cluster heads and provides top-level guidance.
- *Cluster head*. This coordinates lower level controllers and propagates guidance from the root to lower layers.
- *Leaf*. This performs low-level tasks and executes commands coming from the upper layers.

In this chapter, we provide a Petri net plant model for each level of each hierarchy. The Petri net models of the hierarchies can be found in Appendix 51A.

We have identified numerous global consistency issues in the system that require a controller to constrain the actions taken by the hierarchies. These requirements were captured as control specifications and are used to derive the appropriate control structures.

Figure 51.2 shows the hierarchical relationship between the three node levels. To make the hierarchy adaptive, a cluster head can control any number of leaves. Similarly, a root node can coordinate an arbitrary number of cluster heads.

Although there are three tiers within the network hierarchy design, the design does not limit the physical network to only three levels. Networks which are intended to cover a large physical area, or to operate in a highly cluttered environment, may require more nodes than can be effectively managed by three tiers. For this reason it is desirable to allow recursion within the hierarchy. Internal nodes can be inserted between the root node and cluster heads. Internal nodes are implemented by defining either root or cluster head nodes so that they can be connected recursively. This allows complex structures to arise as required by the mission. Figure 51.3 shows such a simple example.

In the network communication and collaborative sensing hierarchies, the root nodes are recursive. For example, in the network communication hierarchy the root node's activities can be described in terms of interactions with a supervisor and data collection from a subnet. The root node expects from the subnet supervisor a set of data providing statistics on each area covered. Similarly, the root node reports to the subnet supervisor a set of data about the network or subnetwork that is being supervised by the root. In this manner, a communication network may in fact contain four or more levels. A network containing four levels would consist of a number of three-level subnets, each supervised by a root node. These root nodes at the third tier would each, in turn, report subnet statistics to an overseeing ''master'' root at the fourth tier. The master root would manage each of the three-level subnets according to subnet capacities. In other words, collections of cluster heads are subnets controlled by a root node. Combinations of cluster heads and root nodes can be controlled by another root node. In this manner, the network may be expanded to manage an arbitrary level of complexity.

Recursion in the network communication and collaborative sensing hierarchies takes place at the root node; however, for the command-and-control hierarchy, recursion takes place at the cluster head.

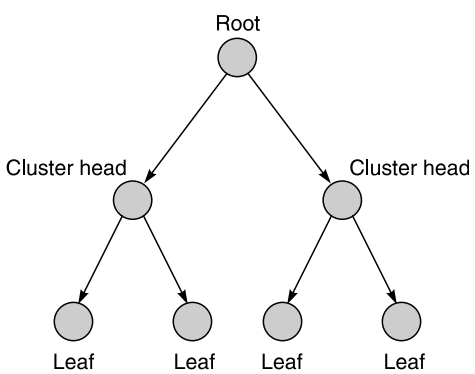


Figure 51.2. Relationships between three node levels.

© 2005 by Chapman & Hall/CRC

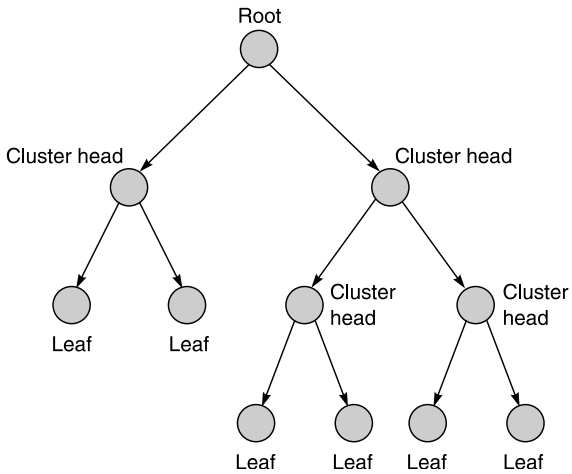


Figure 51.3. Example of a more complex structure.

As discussed previously, the network communication and collaborative sensing network hierarchies are designed in a fashion in which supervising nodes at each level oversee the activities of subnets. This differs from the operational command hierarchy, where the top level of the hierarchy must be designed as a supervisor overseeing the network as opposed to a subnet. The mapping functions and the topology maintenance require that specific methods be implemented at the tier charged with overseeing the entire network. For this reason, the recursion in the operational command hierarchy is implemented at the cluster head level, the highest level in the hierarchy based on a supervisor–subnet philosophy. The root node controls a set of cluster heads. Cluster heads can coordinate leaf nodes and/or other cluster heads. The independent design and implementation allows recursion in different hierarchies to be designed at different tiers without complications.

A given physical node will have a ''rank'' or ''level'' in each of the three hierarchies mentioned. It is important to note that a node's position in one hierarchy is completely independent of its ranking in the other two hierarchies (e.g. a node could be a root in the communication hierarchy, a cluster head in the command-and-control hierarchy, and a leaf in the collaborative sensing hierarchy. This allows for maximum flexibility in terms of network configuration, as well as allowing the network the ability to configure the sensing clusters dynamically to best process information concerning an individual target event occurrence.

51.3.2 Operational Command

The combined operational command hierarchy controls allocation of nodes to surveillance regions, including mapping unknown territory and discovering obstacles. It also controls node deployment and decisions to recall nodes. Figure 51A.1, (see Appendix 51A.5) demonstrates the interaction between the root, cluster heads, and leaf nodes.

The network reconfigures itself as priorities change. Initial node deployments are likely to concentrate nodes in the following regions: (i) where it is assumed enemy traffic will be heavy; (ii) which are of strategic interest to friendly forces. Over time the network should find the areas where enemy traffic is actually flowing, which are likely to be different than initially anticipated. In a similar manner, the strategies of friendly forces are likely to change over time.

The root node manages network resources and oversees the following network functions: mapping the region of interest, node assignment, node reallocation, network topology, and network recall. The root provides information about these functions to the end user and distributes user preferences and commands to appropriate subnets. A pictorial description of the root node is provided in the upper portion of Figure 51A.1.

© 2005 by Chapman & Hall/CRC

Cluster heads (Figure 51A.1, middle) manage the activities of subnets of leaf nodes and other cluster heads, generate topology reports, interpret commands from the root, calculate resource needs, and monitor resource availability.

Leaf node (Figure 51A.1, bottom) responsibilities are limited to only a small portion of the total area being covered by the entire network. These nodes only consider the area they are currently monitoring and retain no global information. Each leaf node directly interacts with its environment, performing terrain mapping and providing position and status information as required by upper levels of the hierarchy.

51.3.3 Network Communications

The network communications hierarchy is implemented to maintain data flow in the presence of environmental interference, such as jamming and node loss. Actions the hierarchy controls include adjusting transmission power, frequency-hopping schedules, *ad hoc* routing, and movement to correct interference. The combined Petri net models in Figure 51A.3 (see Appendix) describe how and when these actions are taken.

The Petri net hierarchy describes a communications protocol between the nodes. Critical messages have associated acknowledgements. To ensure connectivity between nodes and their immediate superiors, all messages passing information up the hierarchy have matching acknowledgements. If an acknowledgement is not received, then retransmission occurs according to parameters set by end users. When retransmissions are exhausted, a supervisor may have to be replaced. When communications with their supervisor are severed, leaf nodes (Figure 51A.3, bottom) and cluster head nodes (Figure 51A.3, middle) immediately enter a promotion cycle. The node waits for an indication that a replacement supervisor has been chosen. If none is received, then the node promotes itself to the next level. It broadcasts that it has assumed control of the subnet and takes over supervisory responsibility. If the previous supervisor rejoins the subnet, then it may demote itself.

Lost contact between the root node (Figure 51A.3, top) and the user is more difficult to address. Upon exhausting retransmissions, the root assumes contact has been lost and it is isolated from the network. The first action taken is to broadcast a message throughout the network indicating to the user that root contact has been lost. Each node tries to establish contact with the user and become the new root. If this fails, the network is put to sleep by a command propagated down the hierarchy. At this point it is left to the user to re-establish contact. While in this quiescent mode the network suspends operations, and responds only to a wake command transmitted by a member of the user community.

51.3.4 Collaborative Sensing

Coordination of sensor data interpretation is done using the collaborative sensing hierarchy shown in Figure 51A.2 (see Appendix). This hierarchy design is based partly on our existing sensor network implementation, which was tested at 29 Palms Marine Base in November 2001.

Initial processing of sensor information is done by the leaf node (Figure 51A.2, bottom). Time series data are preprocessed. A median filter reduces white noise and a low-pass filter removes high-frequency noise. If the signal is still unusable, then it is assumed either that the sensor is broken or that environmental conditions make it impossible, and thus the node temporarily hibernates to save energy.

Each node has multiple sensors and may have multiple sensing modalities, reducing the node's vulnerability to mechanical failure of the sensors and many types of environmental noise [5].

After filtering, sensor time series are registered to a common coordinate system and given a time stamp. Subsequently, data association determines which detections refer to the same object. A state vector with inputs from multiple sensing modalities can be used for target classification [6]. Each leaf node can send either a target state vector or closest point of approach event to the cluster head. A cluster head is selected dynamically. Cluster heads (Figure 51A.2, middle) take care of combining these statistics into meaningful track information.

© 2005 by Chapman & Hall/CRC

Root nodes (Figure 51A.2, top) coordinate activities among cluster heads and follow tracks traversing the area they survey. In this hierarchy, internal nodes are root nodes. They define the sensing topology, which organizes itself from the bottom up. This topology mimics the flow of targets through the system. It has been suggested that this information can guide future node deployment [7].

Sensing hierarchy topology can be calculated using computational geometry and graph theory. A root node can request topology data from all nodes beneath it. Voronoi diagrams are constructed given the locations of nodes. Maximal breach paths and covered paths can be calculated in this region. These data define the system topology and the quality of service (surveillance) [8].

51.4 Control Specifications

Given the set of states G and the set of events Σ, the controller disables a subset of Σ as necessary at every state $g \in G$. Control specifications are defined by identifying state and event combinations that lead the system to an undesirable state. Each specification is a constraint on the system and the controller's behavior is defined by the set of constraints. Control of the DSN requires coordination of individual node activities within the constraints of mission goals. Each node has a set of responsibilities and must act according to its capabilities in response.

The controller is needed because the system has multiple command hierarchies. Each hierarchy has its own goals. When conflicts between hierarchies arise, the controller resolves them. We identified sequences of events that lead to undesirable states. Three primary issues were found that can cause undesirable system states: (i) movement of a node conflicting with the needs of another hierarchy; (ii) nodes attempting to function in the presence of unrecoverable noise; (iii) retreat commands from the command hierarchy should have precedence over all other commands. The following is the set of constraints the controllers impose on the DSN:

CC — operational command
SC — collaborative sensing
WC — network communication

1. When a node is waiting for on-board data fusion it should be prevented from moving by WC, CC and SC. Also, it should not be promoted by WC or by SC until sensing is complete.
2. Hibernation induced by unrecoverable noise or saturated signal in SC should also force the node to hibernate in WC and CC. (And *vice versa*, for leaf nodes only.) Wake-up in SC needs to send wake-up to CC/WC
3. While the cluster head is in the process of updating its statistics, its leaves should be prevented from moving by WC, CC, or SC.
4. While a cluster head node is receiving statistics from its leaf nodes, it should be prevented from moving by WC, CC, or SC.
5. When sensor nodes are in low-power mode as determined by WC, or in damaged mode as determined by CC, they should be prohibited from any moving for prioritized relocation or occlusion adjustments.
6. Retreat in CC should supercede all actions, except propagation of retreat command.
7. Nodes encountering a target signal in the SC should suspend mapping action in CC until sensing is complete.
8. Move commands in CC/WC should be delayed while the node is receiving sensing statistics from lower levels in the hierarchy.

51.5 Controller Design

Each controller design method enforces constraints in its own way. Vector controllers use state vector comparison to determine the transitions that violate the control specifications. Petri net controllers use

© 2005 by Chapman & Hall/CRC

slack variables to disable the same transitions. Moore machines determine which strings of events lead to constraint violations.

Controller design is complicated by the existence of uncontrollable and unobservable transitions. Uncontrollable transitions cannot be disabled; unobservable transitions cannot be detected. When uncontrollable or unobservable transitions lead to undesirable states, the controller design process requires creating alternative constraints that use only controllable transitions. Ideally, the controller should not unnecessarily constrain the system. One particular methodology for creating nonrestrictive controllers is described by Moody [9].

Control specification is usually specified as $l\mu \leq b$, where l is an $N \times M$ matrix (number of control specifications by the number of places in the plant), μ is an $M \times 1$ matrix representing number of tokens in each place of the plant, and b is an $N \times 1$ integer matrix with each element representing the total maximal allowed number of tokens in any combination of places.

51.5.1 FSM Controller

Verifying system properties, such as safeness, boundedness, and liveness, is done using the Karp–Miller tree. It represents all possible states of the system. Figure 51.4 shows a Petri net example and its associated Karp–Miller Tree [4].

The following is the Karp–Miller algorithm [10]:

1. Label initial marking S_0 as the root of the tree and tag it as new.
2. While new markings exist do:
   - 2.1. Select a marking S.
   - 2.2. If S is identical to a marking on the path from the root to S, then tag S as old and go to another marking.
   - 2.3. If no transitions are enabled at S, tag S dead-end.
   - 2.4. While there exist enabled transitions at S do:
     - 2.4.1. Obtain the marking S' that results from firing T at S.
     - 2.4.2. On the path from the root to S, if there exists a marking S'' such that $S'(p) \geq S''(p)$ for each place p and S' is different from S'', then replace $S'(p)$ by ω for each p such that $S'(p) > S''(p)$.
     - 2.4.3. Introduce S' as a node, draw an arc with label T from S to S' and tag S' as new.

Ramadge and Wonham [2] described the supervisory control of a discrete event process using a finite state automaton. We generalized their contribution and proposed our own innovations.

All reachable state vectors could be infinite, but the Karp–Miller tree should be finite. Thus, we introduce the symbol ω in the Karp–Miller tree to indicate that the token number in the corresponding place is unbounded. A 5-tuple plant $\wp = (Q, \Sigma, \delta, q_0, Q_m)$ was obtained from the Karp–Miller tree,

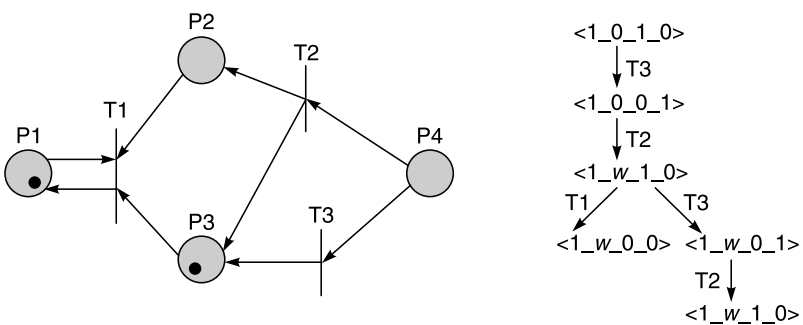


Figure 51.4. A sample Petri net (left) and its associated Karp–Miller tree (right).

© 2005 by Chapman & Hall/CRC

where Q is all legal and illegal states, Σ is all transitions, δ is the next state function, q_0 is the initial state, and Q_m is only the legal states. Because the FSM generated without constraints contains illegal states, we enforce a state feedback map function on the plant to restrict its behavior. Let $\Gamma = (0,1)^{\Sigma_c}$ be a set of control patterns. For each $\gamma \in \Gamma$, γ: $\Sigma_c \rightarrow \{0, 1\}$ is a control pattern of $|\Sigma_c|$ bits. An event σ is enabled if $\gamma(\sigma) = 1$. For uncontrollable transitions, $\gamma(\sigma)$ always equals one. Then, we define an augmented transition function as

$$\delta_c : \Gamma \times \Sigma \times Q \rightarrow Q \tag{51.1}$$

according to:

$$\delta_c(\gamma,\sigma,q) = \begin{cases} \delta(\sigma, q) & \text{if } \delta(\sigma, q) \text{ is defined and } \gamma(\sigma) = 1 \\ \text{undefined} & \text{otherwise} \end{cases} \tag{51.2}$$

We interpret this controlled plant as $\wp_c = (Q, \Gamma \times \Sigma, \delta_c, q_0, Q_m)$, which admits external control [2].

The Moore machine is a 5-tuple, represented as $(S, I, O, \delta, \Gamma)$, where S is the nonempty finite set of states, I is the nonempty finite set of inputs, O is the nonempty finite set of outputs, δ is the next state function, which maps $S \times I \rightarrow S$, and Γ is the output function, which maps $S \rightarrow O$. The state feedback map can be realized by the output function of the Moore machine, which defines a mapping between the current state and a control pattern for the current state.

Ramadge and Wonham [2] acquire the state feedback map by enumerating all legal states in the FSM together with their binary control patterns. Introducing the Moore machine and state encoding automatically yields the control pattern from derived logical expressions in terms of their current state.

First, we trim the Karp–Miller tree to reach a finite state automaton as a recognizer for the legal language of the plant. $\lceil \log_2 N \rceil$ bits are then used to encode N legal states. Since the choice of encoding affects the complexity of logic implementation, an optimal encoding strategy is preferred. The transition table is used to derive logical expressions in terms of binary encoded state for each controllable transition. State minimization is carried out to remove redundant states [11].

This approach to an FSM modeled controller is unique in two respects. Instead of exploring the algebraic or structural property of a Petri net, as in the case of a vector discrete event system (VDES) and Petri net controllers, it utilizes traditional finite automata to tackle the control problem of a discrete event system. In addition, the introduction of the Moore machine to output controller variables guarantees a real-time response. The quick response is acquired at the cost of extensive searching and filtering of the entire reachable state space offline.

The FSM modeled controllers perform well for small- and medium-scale systems, but representation and computation costs would be prohibitively expensive for complex systems. One alternative is to model the system with Petri nets. The current Petri net state vector is converted to a binary encoded state and then a binary control pattern is calculated. Overhead is incurred while converting the state vector to binary encoded form, but the representative power of Petri nets is greater than that of an FSM. Also, instead of the traditional brute-force search of the entire state space, we examine only those transitions that have an elevated effect on the right-hand side of our control specifications. All transitions are screened and only those that would result in an increase in the left-hand side of the control specification ($l\mu \leq b$) as described in Section 51.5 are candidates for control. The binary control pattern bit for a particular transition is set to one when $l\mu \leq b$ continues to hold after the transition firing. For multiple control specifications, the binary control pattern for a particular transition is one if and only if the current state satisfies the conjunction of all the inequalities imposed by all constraints. In this case, the binary control pattern is software determined instead of hardware determined. The sample controller for our DSN can be found in Section 51.5.4.1.

© 2005 by Chapman & Hall/CRC

51.5.2 Vector Addition Controller

The VDES approach represents the system state as an integer vector. State transitions are represented by integer vector addition [12]. The VDES is an automaton that generates a language over a finite alphabet Σ consisting of two subsets: Σ_c and Σ_{uc}. Σ_c is the set of controllable events that can be disabled by the external controller. Σ_{uc} is the set of uncontrollable events that cannot be disabled by the controller. We use the following symbols: $G_{uc} \in G$ is the uncontrollable part of the plant G; D is the incidence matrix of the plant constructed as by David and Alta [3] (places are rows; transitions are columns; $x_{ij} = 1$ (-1) if an arc leads from place i to transition j, else $x_{ij} = 0$); D_{uc} is the uncontrollable transition columns of the incidence matrix; D_{uo} is the unobservable transition columns of the incidence matrix; Σ is all transitions in the plant; $\Sigma_{uc} \in \Sigma$ is the subset of transitions that are uncontrollable; $\Sigma_{uo} \in \Sigma$ is the subset of transitions that are unobservable $L(G, \mu)$ is the language of the plant starting with marking μ (i.e. the set of all possible sequences of transitions; the language can be directly inferred from the Karp–Miller tree, which we show how to compute in Section 51.5.1); and $\omega \in L(G, \mu)$ is a valid sequence of transitions in the plant starting from the state μ.

Given a Petri net with incidence matrix D and a control specification $l\mu \leq b$, a final state can be represented as a single vector equation as follows. Given a sequence of N events, $\omega \in L(G, \mu)$, the final state μ_N is given by

$$\mu_N = \mu_0 + Dq_1 + Dq_2 + \cdots + Dq_N = \mu_0 + D(q_1 + q_2 + \cdots q_N) = \mu_0 + DQ_\omega \tag{51.3}$$

$Q_{\omega(i)} = |\omega|_{q_i}$ represents the number of occurrences of q_i in the event sequence. The number of event occurrences, independent of how the events are interleaved, thus defines the final state. We use the following Boolean equation:

$$f^*(\alpha,\mu) = \begin{cases} 1 & \text{if } \mu + Dq_\alpha \in [P] \\ 0 & \text{else} \end{cases} \tag{51.4}$$

where

$$[P] = \left\{\mu \middle| (\forall \omega \in L(G_{uc}, \mu)), l\left(\mu + D_{uc}Q_{uc,\omega}\right) \leq b\right\} = \left\{\mu \middle| \left(l\mu + \max_{\omega \in L(G_{uc},\mu)} lD_{uc}Q_{uc,\omega}\right) \leq b\right\} \tag{51.5}$$

The transition associated with q_α is allowed to fire only if no subsequent firing of uncontrollable transitions would violate the control specification $l\mu \leq b$.

In general, the maximization problem in Equation (51.5) is a nonlinear program with an unstructured feasible set $L(G_{uc}, \mu)$. However, a theorem proven by Li and Wonham [13] shows that when G is loop free, for every state where

$$\mu \geq 0 \text{ and } Q \geq 0, \quad \mu + DQ \geq 0 \Leftrightarrow (\exists \omega \in L(G, \mu))Q = Q_\omega \tag{51.6}$$

the computation of $[P]$ can be reduced to a linear integer program. The set of possible strings $\omega \in L(G, \mu)$ can then be simplified as

$$\left\{Q_{uc,\omega} \middle| \omega \in L(G_{uc}, \mu)\right\} = \left\{Q \in Z^K | \mu + D_{uc}Q \geq 0, Q \geq 0\right\} \tag{51.7}$$

With this simplification of the feasible region, the set $[P]$ of allowed states becomes

$$[P] = \left\{\mu \middle| l\mu + lD_{uc}Q^*(\mu) \leq b\right\} \tag{51.8}$$

© 2005 by Chapman & Hall/CRC

where $Q^*(\mu)$ is the solution for

$$\max_{Q} lD_{uc}Q \quad \text{s.t.} \begin{cases} D_{uc}Q \geq -\mu \\ Q \geq 0 \,(\text{int}) \end{cases} \tag{51.9}$$

yielding Q^* as a function of μ [14].

To confirm the controllability, it suffices to test whether or not the initial marking of the system satisfies the equation

$$\mu_0 \epsilon [P] \quad \text{or} \quad \left(l\mu_0 + \max_{\omega \in L(G_{uc}, \mu_0)} lD_{uc}Q_{uc,\omega} \right) \leq b \tag{51.10}$$

If Equation (51.10) is not satisfied, then no controller exists for this control specification [13].

When illegal markings are reachable from the initial marking by passing through a sequence of uncontrollable events, it is an inadmissible specification. Inadmissible control specifications must take an admissible form before synthesizing a controller. Equation (51.5) is the transformed admissible control specification.

Essentially, a VDES controller is the same as a Petri-net-modeled controller. A controller variable c is introduced into the system as a place with the initial value to be b minus the initial value of the transformed admissible control specification [12]. A controllable event will be disabled if and only if its occurrence will make c negative. In our implementation, the controller examines all enabled controllable transitions. If the firing of a transition leads to an illegal state, then the system rolls back and continues looking for the next enabled transition.

51.5.3 Petri-Net-Based Control

Li and Wonham [12,13] made significant contributions to the control of plants with uncontrollable events by specifying conditions under which control constraint transformations have a closed-form expression. However, the loop-free structure of the uncontrollable subplant is a sufficient but not necessary condition for control. Moody [15] extended the scope of controller synthesis problems to include unobservable events, in addition to uncontrollable events already discussed in VDES. He also found a method for controller synthesis for plants with loops containing uncontrollable events.

In the Petri net controller, a plant with n places and m transitions has incidence matrix $D_p \in Z^{n \times m}$. The controller is a Petri net with incidence matrix $D_c \in Z^{n_c \times m}$. The controller Petri net contains all the plant transitions and a set of control places. Control places are used to control the firing of transitions when control specifications will be violated. Control places cannot have arcs incident on unobservable or uncontrollable transitions. Arcs from uncontrollable transitions to control places are permitted. As with VDES, inadmissible control specifications must be converted to admissible control specifications before controller synthesis. An invariant-based control specification $l\mu \leq b$ is admissible if $lD_{uc} \leq 0$ and $lD_{uo} = 0$.

If the original set of control specifications $L\mu \leq b$ contains inadmissible specifications, then it is necessary to define an equivalent set of admissible specifications. Before proceeding with this step, we need to prove that the state space of the new control specifications lies within the state space of the original control specifications. Let $R_1 \in Z^{n_c \times n}$ satisfy $R_1\mu \geq 0 \forall \mu$. Let $R_2 \in Z^{n_c \times n_c}$ be a positive definite diagonal matrix. If

$$L'\mu \leq b' \quad \text{where} \quad \begin{aligned} L' &= R_1 + R_2 L \\ b' &= R_2(b+1) - \mathbf{1} \end{aligned} \tag{51.11}$$

$\mathbf{1}$ is an n_c-dimensional vector of 1's, then $L\mu \leq b$. The proof is given by Moody and Antsaklis [15].

© 2005 by Chapman & Hall/CRC

To construct a controller that does not require inhibiting uncontrollable transition or detecting unobservable transitions, it is sufficient to calculate two matrices R_1 and R_2 which satisfy

$$[R_1 R_2]\begin{bmatrix} D_{uc} & D_{uo} & -D_{uo} & \mu_0 \\ LD_{uc} & LD_{uo} & -LD_{uo} & L\mu_0 - b - \mathbf{1} \end{bmatrix} \leq [0\ 0\ 0 - 1] \tag{51.12}$$

The first column in Equation (51.12) indicates that $LD_{uc} \leq 0$; the second and the third columns indicate that $LD_{uo} = 0$; and the fourth column indicates that the initial marking of the Petri net satisfies the newly transformed admissible control specification. Using the admissible control specification, a slack variable μ_c is introduced to transform the inequality into an equality:

$$L'\mu + \mu_c = b' \tag{51.13}$$

Thus

$$\begin{aligned} D_c &= -(R_1 + R_2 L)D_p = -L'D_p \\ \mu_{c0} &= R_2(b + \mathbf{1}) - \mathbf{1} - (R_1 + R_2 L)\mu_0 = b' - L'\mu_0 \end{aligned} \tag{51.14}$$

Equation (51.14) provides the controller incidence matrix and initial marking of the control places.

In contrast, with the VDES controller a Petri net controller explores the solution by inspecting the incidence matrix. Plant/controller Petri nets provide a straightforward representation of the relationship between the controller and controlled components. The evolution of the Petri net plant/controller is easy to compute, which facilitates usage in real-time control problems. In our implementation, the plant/controller Petri net incidence matrix is the output that results from the plant and control specification as input [15].

51.5.4 Performance and Comparison of Three Controllers

Figure 51.5 is an example of a Petri net consisting of two independent parts with three uncontrollable transitions: T2, a T3, and T5 with an initial marking of $[2\ 0\ 0\ 0\ 1\ 1\ 0]^T$. This net is a reduced form of our DSN. Its main purpose is to illustrate how the control issues are handled in our DSN. Results of the three approaches and comparison are given.

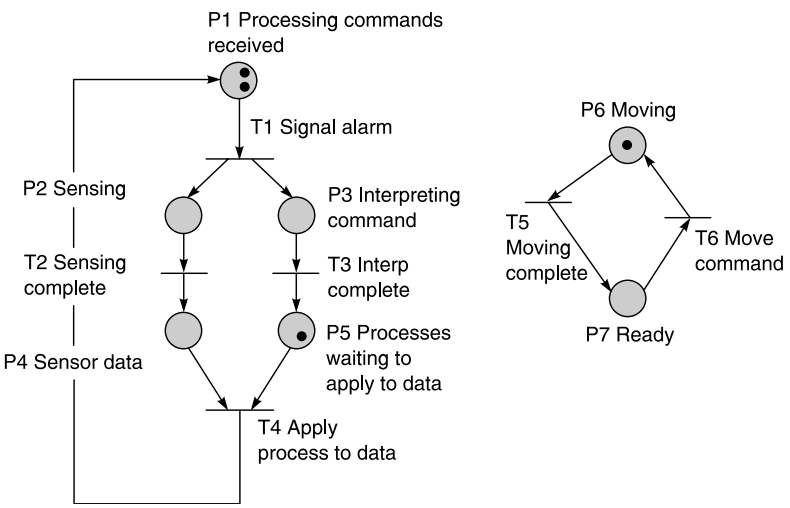


Figure 51.5. A reduced DSN Petri net model.

© 2005 by Chapman & Hall/CRC

The behavior of the two independent Petri nets should obey the control specifications. The first constraint requires that place P5 cannot contain more than two tokens. There cannot be more than two processes active at one time. The second constraint states that the sum of tokens in P2 and P6 must be less than or equal to one. This constraint implies that a node is not allowed to move in the operational command hierarchy, when it is sensing in the scope of the collaborative sensing hierarchy, or *vice versa*. This mutual exclusion constraint represents the major control task of enforcing consistency across independently evolving hierarchies in our DSN. Three uncontrollable transitions are sensing complete, interpreting complete, and moving complete.

Two control specifications are: (1) $\mu_5 \leq 2$; (2) $\mu_2 + \mu_6 \leq 1$.

51.5.4.1 FSM Controller

Detailed steps of how to construct an FSM controller for the reduced DSN model in Figure 51.5 are given. A reachability tree has thus been constructed from a Petri net first. Some of the states that were generated from the plant without constraints are:

States {

s2_0_0_0_1_1_0,	s0_0_0_2_3_0_1,	s1_1_2_0_0_1_0,
s1_1_1_0_1_1_0,	s1_0_0_1_2_0_1,	s0_2_3_0_0_1_0,
s0_2_2_0_1_1_0,	s0_1_1_1_2_0_1,	s0_2_3_0_0_0_1,
s0_1_2_1_1_1_0,	s0_0_1_2_2_0_1,	s0_2_2_0_1_0_1,
s0_0_2_2_1_1_0,	s1_0_1_1_1_0_1,	s0_2_1_0_2_0_1,
s0_0_1_2_2_1_0,	s0_1_2_1_1_0_1,	s1_1_2_0_0_0_1,
s0_0_0_2_3_1_0,	s0_0_2_2_1_0_1,	s1_1_1_0_1_0_1,
s1_0_0_1_2_1_0,	s1_0_2_1_0_0_1,	
s0_1_1_1_2_1_0,	s0_1_3_1_0_0_1,	
s0_1_0_1_3_1_0,	s0_0_3_2_0_0_1,	s2_0_1_0_0_0_1,
s1_1_0_0_2_1_0,	s0_0_3_2_0_1_0,	s2_0_0_0_1_0_1
s0_2_1_0_2_1_0,	s0_1_3_1_0_1_0,	}
s0_2_0_0_3_1_0,	s1_0_2_1_0_1_0,	
s0_2_0_0_3_0_1,	s1_0_1_1_1_1_0,	
s0_1_0_1_3_0_1,	s2_0_1_0_0_1_0,	

We search the entire state space removing illegal states, which would either directly or indirectly violate the control specification. A new state space with 13 legal states, as shown below, is achieved. Four bits are needed to encode 13 states as encoded by A, B, C and D. A Moore machine is constructed to output the binary control pattern based on the current encoded state.

State:	Marking	Encode state
State S0:	2000110	0000
State S1:	2000101	0001
State S2:	1110101	0010
State S3:	1011101	0011
State S4:	1001201	0100
State S5:	1001210	0101
State S6:	2010001	0110
State S7:	1120001	0111
State S8:	1021001	1000
State S9:	1021010	1001
State S10:	1011110	1010
State S11:	2010010	1011
State S12:	1100201	1100

Among six transitions, we cannot control T2, T3, or T5. These controllable transitions have firings which would lead to illegal states directly or indirectly. Based on knowledge of offline screening, T1 and

© 2005 by Chapman & Hall/CRC

T6 should be controlled. Thus, the binary control pattern has two bits. The transition table with encoded states for the Moore machine is as follows:

Present state	Next state						Output	
ABCD (4 encoded bits)	T1	T2	T3	T4	T5	T6	for T1	for T6
S0					S1		0	1
S1	S2					S0	1	1
S2		S3	S12				0	0
S3			S4	S6		S10	0	1
S4				S1		S5	0	1
S5				S0	S4		0	1
S6	S7		S1			S11	1	1
S7		S8	S2				0	0
S8			S3			S9	0	1
S9			S10		S8		0	1
S10			S5	S11	S3		0	1
S11			S0		S6		0	1
S12		S4					0	0

From the transition table we can construct a Moore machine state diagram with 13 states, with six inputs and two outputs. The state feedback function, which outputs the binary control patterns based on current state, is used to regulate plant behavior by switching between control patterns.

The logical expression of a binary control pattern can be expressed as

$$\begin{cases} \mathrm{T1} = \overline{\mathrm{A}}\,\overline{\mathrm{B}}\mathrm{C}\overline{\mathrm{D}} + \overline{\mathrm{A}}\mathrm{B}\mathrm{C}\overline{\mathrm{D}} \\ \overline{\mathrm{T6}} = \overline{\mathrm{A}}\mathrm{C}(\overline{\mathrm{B}\mathrm{D}} + \mathrm{B}\mathrm{D}) + \mathrm{A}\mathrm{B}\overline{\mathrm{C}\mathrm{D}} \end{cases}$$

The logical implementation can be realized by hardware. The controller can immediately access the control pattern for each controllable transition based on its current encoded state without going through legal firing checking, as in VDES, or by doing extra calculation involving added controller places and arcs, as in a Petri-net-modeled controller. The trade-off is offline legal state space searching. Following our search method used in the DSN, we simply check whether or not the current state vector satisfies the conjunction of $\mu_3 + \mu_5 \leq 1$ and $\mu_2 + \mu_6 = 0$. If it does, then the control pattern bit of $\mathrm{T1} = 1$, and if $\mu_2 + \mu_6 = 0$, then the control pattern bit of $\mathrm{T6} = 1$. This turns out to be efficient to compute and simple to express for a complex system.

51.5.4.2 VDES Modeled Controller

A VDES controller is formed to control the same reduced DSN model shown in Figure 51.5. D is the incidence matrix of the plant as described in Section 51.5.2. D_{uc} is the uncontrollable portion of D, with rows to be places and columns to be the uncontrollable transitions:

$$D = \begin{bmatrix} -1 & 0 & 0 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} \quad D_{\mathrm{uc}} = \begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \\ 0 & 0 & 1 \end{bmatrix}$$

© 2005 by Chapman & Hall/CRC

The goal of the controller is to enforce linear inequality on the state vector of G, usually in the form of $l\mu \leq b$. Our control specifications are $\mu_5 \leq 2$ and $\mu_2 + \mu_6 \leq 1$. Consider the first control specification:

$$l_1 = [0\ 0\ 0\ 0\ 1\ 0\ 0] \quad b_1 = 2$$

The initial marking satisfies the control specification, but the control specification is inadmissible because the uncontrollable firing of T3 would lead to a violation. Since the uncontrollable part of the system is loop free, the inadmissible control specification can be transformed to an admissible control specification. Solve the $\max_Q lD_{uc}Q$ as discussed in Section 51.5.2:

$$\text{s.t.} \begin{cases} D_{uc}Q \geq -\mu \\ Q \geq 0(\text{int}) \end{cases}$$

By doing this, the effect of uncontrollable events firings on the control specification is taken into consideration.

$$\max_Q l_1 D_{uc} Q = [0\ 0\ 0\ 0\ 1\ 0\ 0] \begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} q_2 \\ q_3 \\ q_5 \end{bmatrix} = \max(q_3)$$

$$\text{s.t.} \begin{cases} \begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} q_2 \\ q_3 \\ q_5 \end{bmatrix} \geq \begin{bmatrix} -\mu_1 \\ -\mu_2 \\ -\mu_3 \\ -\mu_4 \\ -\mu_5 \\ -\mu_6 \\ -\mu_7 \end{bmatrix} \Rightarrow \begin{cases} -\mu_1 \leq 0 \\ \mu_2 \geq q_2 \\ \mu_3 \geq q_3 \\ -\mu_4 \leq q_2 \\ -\mu_5 \leq q_3 \\ \mu_6 \geq q_5 \\ -\mu_7 \leq q_5 \end{cases} \\ q_i \geq 0 \end{cases}$$

From the above, it can be inferred that $\max(q_3) = \mu_3$, as $\mu_3 \geq q_3$. The transformed admissible control specification is $l\mu + \max\ lD_{uc}Q^*(\mu) \leq b$, which is $\mu_5 + \mu_3 \leq 2$. The initial marking $[2\ 0\ 0\ 0\ 1\ 1\ 0]^T$ holds for $\mu_5 + \mu_3 \leq 2$; thus, the controller exists for this control constraint [14]. The second control specification is an admissible one, because no uncontrollable transition firing would lead to an illegal state.

In our controller implementation, the plant, together with two admissible control specifications, is treated as input. The state space of the controlled system is:

2_0_0_0_1_1_0, 1_1_2_0_0_0_1,
2_0_0_0_1_0_1, 1_0_2_1_0_0_1,
1_1_1_0_1_0_1, 1_0_2_1_0_1_0,
1_0_1_1_1_0_1, 1_0_1_1_1_1_0,
1_0_0_1_2_0_1, 2_0_1_0_0_1_0,
1_0_0_1_2_1_0, 1_1_0_0_2_0_1
2_0_1_0_0_0_1,

Each of these states satisfies the two control specifications mentioned previously.

© 2005 by Chapman & Hall/CRC

51.5.4.3 Petri-Net-Modeled Controller

Finally, a Petri net controller is built for the reduced DNS with the same control specifications. We have the same first control specification, where $\mu_5 \leq 2$. Since the plant has no unobservable transitions, we only need to study uncontrollable transitions. The first step is to determine whether the control specification is admissible.

The following indicates an inadmissible control specification as discussed in the second paragraph of Section 51.5.3:

$$[0\ 0\ 0\ 0\ 1\ 0\ 0]D_{\text{uc}} = [0\ 1\ 0] \geq [0\ 0\ 0]$$

It was observed that the third row of D_{uc}, if equal to $[0\ -1\ 0]$, could be used to eliminate the positive element 1 in the above equation. So,

$$R_1 = [0\ 0\ 1\ 0\ 0\ 0\ 0] \quad R_2 = \mathbf{1} \quad L' = R_1 + R_2 L = [0\ 0\ 1\ 0\ 0\ 0\ 0] + 1[0\ 0\ 0\ 0\ 1\ 0\ 0] = [0\ 0\ 1\ 0\ 1\ 0\ 0]$$

The initial marking satisfies the admissible control specification. The transformed admissible control specification is $\mu_5 + \mu_3 \leq 2$, which is the same as the admissible control specification from the above section. By introducing a new slack variable μ_{c}, the control specification becomes an equation:

$$\mu_5 + \mu_3 + \mu_c = 2$$

For the second admissible control specification, $\mu_2 + \mu_6 \leq 1$, we introduce another slack variable μ'_{c}, and the second control specification becomes another equation: $\mu_2 + \mu_6 + \mu'_{\text{c}} = 1$.

$$D = \begin{bmatrix} -1 & 0 & 0 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} \quad \mu_0 = [2\ \ 0\ \ 0\ \ 0\ \ 1\ \ 1\ \ 0]^{\text{T}}$$

$$L = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \quad b = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$D_{\text{c}} = -LD = \begin{bmatrix} -1 & 0 & 0 & 1 & 0 & 0 \\ -1 & 1 & 0 & 0 & 1 & -1 \end{bmatrix} \quad \text{and} \quad \mu_{\text{c0}} = b - L\mu_0 = \begin{bmatrix} 2 \\ 1 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

The resulting overall plant/controller incidence matrix is

$$D_{\text{all}} = \begin{bmatrix} -1 & 0 & 0 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 1 & -1 \\ -1 & 0 & 0 & 1 & 0 & 0 \\ -1 & 1 & 0 & 0 & 1 & -1 \end{bmatrix}$$

© 2005 by Chapman & Hall/CRC

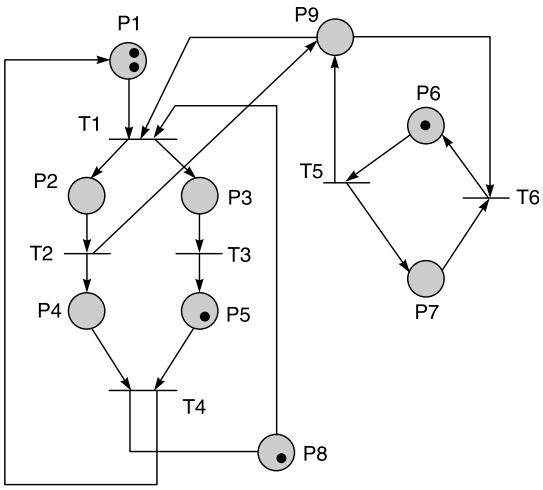


Figure 51.6. Plant/controller Petri net model.

Two controller places can be added to the plant as P8 and P9 with initial marking of one and zero respectively, as shown in Figure 51.6. In our implementation, the Petri-net-modeled controller implementation computes a closed-loop Petri net incidence matrix based on the plant and the control constraints to be enforced without going through the above manual computation.

Running our implementation program, we get an FSM as shown below:

```
states {
s2_0_0_0_1_1_0_1_0,
s2_0_0_0_1_0_1_1_1,
s1_1_1_0_1_0_1_0_0,
s1_0_1_1_1_0_1_0_1,
s1_0_0_1_2_0_1_0_1,
s1_0_0_1_2_1_0_0_0,
s2_0_1_0_0_0_1_1_1,
s1_1_2_0_0_0_1_0_0,
s1_0_2_1_0_0_1_0_1,
s1_0_2_1_0_1_0_0_0,
s1_0_1_1_1_1_0_0_0,
s2_0_1_0_0_1_0_1_0,
s1_1_0_0_2_0_1_0_0}
}
transitions {
<s2_0_0_0_1_1_0_1_0,t5,s2_0_0_0_1_0_1_1_1,>,
<s2_0_0_0_1_0_1_1_1,t1,s1_1_1_0_1_0_1_0_0,>,
<s1_1_1_0_1_0_1_0_0,t2,s1_0_1_1_1_0_1_0_1,>,
<s1_0_1_1_1_0_1_0_1,t3,s1_0_0_1_2_0_1_0_1,>,
<s1_0_0_1_2_0_1_0_1,t4,s2_0_0_0_1_0_1_1_1,>,
<s1_0_0_1_2_0_1_0_1,t6,s1_0_0_1_2_1_0_0_0,>,
<s1_0_0_1_2_1_0_0_0,t4,s2_0_0_0_1_1_0_1_0,>,
<s1_0_0_1_2_1_0_0_0,t5,s1_0_0_1_2_0_1_0_1,>,
<s1_0_1_1_1_0_1_0_1,t4,s2_0_1_0_0_0_1_1_1,>,
<s2_0_1_0_0_0_1_1_1,t1,s1_1_2_0_0_0_1_0_0,>,
<s1_1_2_0_0_0_1_0_0,t2,s1_0_2_1_0_0_1_0_1,>,
<s1_0_2_1_0_0_1_0_1,t3,s1_0_1_1_1_0_1_0_1,>,
<s1_0_2_1_0_0_1_0_1,t6,s1_0_2_1_0_1_0_0_0,>,
<s1_0_2_1_0_1_0_0_0,t3,s1_0_1_1_1_1_0_0_0,>,
<s1_0_1_1_1_1_0_0_0,t3,s1_0_0_1_2_1_0_0_0,>,
<s1_0_1_1_1_1_0_0_0,t4,s2_0_1_0_0_1_0_1_0,>,
<s2_0_1_0_0_1_0_1_0,t3,s2_0_0_0_1_1_0_1_0,>,
<s2_0_1_0_0_1_0_1_0,t5,s2_0_1_0_0_0_1_1_1,>,
<s1_0_1_1_1_1_0_0_0,t5,s1_0_1_1_1_0_1_0_1,>,
<s1_0_2_1_0_1_0_0_0,t5,s1_0_2_1_0_0_1_0_1,>,
<s1_1_2_0_0_0_1_0_0,t3,s1_1_1_0_1_0_1_0_0,>,
<s2_0_1_0_0_0_1_1_1,t3,s2_0_0_0_1_0_1_1_1,>,
<s2_0_1_0_0_0_1_1_1,t6,s2_0_1_0_0_1_0_1_0,>,
<s1_0_1_1_1_0_1_0_1,t6,s1_0_1_1_1_1_0_0_0,>,
<s1_1_1_0_1_0_1_0_0,t3,s1_1_0_0_2_0_1_0_0,>,
<s1_1_0_0_2_0_1_0_0,t2,s1_0_0_1_2_0_1_0_1,>,
<s2_0_0_0_1_0_1_1_1,t6,s2_0_0_0_1_1_0_1_0,>
}
   inputs {
   +t1 +t2 +t3 +t4 +t5 +t6
   }
                                  outputs {
                                  }
```

All these states derived from the program are legal.

51.6 Case Study

51.6.1 Simulation Result

Software was developed to simulate the actions of a DSN represented by a Petri net plant model. The constraints listed in Section 51.4 were those to be monitored and enforced by each of the

© 2005 by Chapman & Hall/CRC

controllers. The Petri net plant model of the DSN consisted of 133 places, 234 transitions, and roughly 1000 arcs. In order to enforce the plain-language constraints, 44 inequalities of the form $l\mu \leq b$ were generated.

The Petri net controller was implemented automatically by creating 44 control places that would act as the slack variables in a closed-loop Petri net. Arcs from these controller places influence controllable transitions in the plant net in an effort to enforce the constraints. Thus, the controlled plant Petri net is simply a new Petri net with additional places and arcs. Unlike the Petri net controller, the VDES controller required no additional places or arcs to control the plant net. The VDES controller was implemented by examining every possible enabled firing given a plant state. The controller then examined the state of the system should each of these enabled firings take place, and disabled those transitions whose firings led to a forbidden state. This characteristic of VDES control illustrates a similarity with Moore machines; in Moore machines, the entire state space is explored offline and all forbidden strings are known *a priori*. In the case of VDES, exploration of reachable states is undertaken at each state dynamically and is limited to those states directly reachable from the current state.

The plant model was activated and the set of forbidden states was monitored at each transition firing. Without a controller of any kind in place, the plant model reached a forbidden state in less than 10,000 transition firings in each test. When the Petri net or the VDES controllers were implemented, the plant model ran through 100,000 transition firings without violation. Thus, each controller was found to be effective in preventing the violation of system constraints, and the choice of which to use can be based upon issues such as execution speed. It was found that the relationship between the initial state and the controller specification was crucial. In complex systems, such as the DSN, it is not difficult to specify an initial marking that will make the plant uncontrollable. Care must be taken to ensure that the system design and marking do not render the controller useless.

51.7 Discussion and Conclusions

Faced with the problem of synthesizing a controller for our large-scale surveillance network, we selected three methods as candidates and applied them to our system. Through comparison, we concluded that the approaches are roughly equivalent, each with pros and cons. Generally speaking, the three approaches can be classified into two categories: FSM belongs to traditional finite-automata-based controller category and Petri-net-modeled and VDES belongs to the Petri-net-based controller family.

The traditional Ramadge and Wonham [2] control model is based on a classic finite automaton. Unfortunately, FSM-based controllers involve exhaustive searches or simulation of system behavior and are especially impractical for large and complex systems. We eliminate illegal state spaces before synthesizing our finite automata, but the process is still a computationally expensive process for a system with a large number of states and events. Offline searching of the entire set of reachable states and the hardware implementation of the logical expression assures prompt controller response, which is crucial for those systems with strict real-time requirements. For a complex system, such as a surveillance system, we use a modified version of an FSM-modeled controller to avoid expensive computation and high representation cost. The controller is directly derived from the control specifications.

On the contrary, Petri-net-based controllers take full advantage of the properties of the Petri net. Their efficient mathematical computation, employing linear matrix algebra, makes real-time controlling and analysis possible, but they are still inferior to an FSM in the performance of response time. Petri nets offer a much more compact state space than finite automata and are better suited to model systems that exhibit a repetitive structure. Automatic handling of concurrent events is maintained, as shown by Wonham [14] and Moody and Antsaklis [15]. VDES controllers explore the maximally permissive control constraint on the Petri net with uncontrollable transitions by application of the integer linear

© 2005 by Chapman & Hall/CRC

programming problem, assuming that the uncontrollable portion of the Petri net has no loops and the actual controller exists [15]. However, VDES does not consider unobservable events. The loop-free condition proves to be a sufficient, but not a necessary condition. Petri-net-modeled controllers investigate the structural properties of a controlled Petri net with unobservable events in addition to uncontrollable events. The integrated graphical structure of the Petri net plant/controller makes system computation and representation straightforward.

The simulation results show that the system behavior is similarly and effectively constrained by any of the three approaches. Secondary concerns, such as execution time and ease of representation, can therefore guide the decision on which approach to use.

Acknowledgments and Disclaimer

This material is based upon work supported by the U.S. Army Robert Morris Acquisition under Award No. DAAD19-01-1-0504. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the army.

Reference

[1] Bulusu, N. *et al.*, Scalable coordination for wireless sensor networks: self-configuring localization systems, USC/Information Sciences Institute, 2001.
[2] Ramadge, P.J. and Wonham, W.M., Supervisory control of a class of discrete event progress, *SIAM Journal on Control and Optimization*, 25(1), 206, 1987.
[3] David, R. and Alla, H., Petri Nets and Grafcet Tools for Modeling Discrete Event Systems, Prentice Hall, 1992.
[4] Peterson, J.L., Petri nets, *Computing Surveys*, 9(3), 223, 1977.
[5] Brooks, R. and Iyengar, S.S., *Multi Sensor Fusion: Fundamentals and Applications with Software*, Prentice Hall, New Jersey, 1997.
[6] Luo, R.C. and Kay, M.G., Multisensor integration and fusion in intelligent systems, *IEEE Transactions on Systems, Man, and Cybernetics*, 19(5), 901, 1989.
[7] Deb, B. *et al.*, A topology discovery algorithm for sensor networks with applications to network management, Technical Report DCS-TR-441, Department of Computer Science, Rutgers University, May 2001, IEEE CAS workshop, September 2002.
[8] Meguerdichian, S. *et al.*, Coverage problems in wireless *ad-hoc* sensor networks, Computer Science Department, Electrical Engineering Department, University of California, Los Angeles, May 2000.
[9] Moody, J.O., Petri net supervisors for discrete event systems, Ph.D. Dissertation, Department of Electrical Engineering, Notre Dame University, April 1998.
[10] http://www-cad.eecs.berkeley.edu/~polis/class/ee249/lectures/lec06.pdf (last accessed on 7/26/2004).
[11] Aho, A.V. *et al.*, *Compilers: Principles, Techniques and Tools*, Addison-Wesley, Reading, MA, 1986.
[12] Li, Y. and Wonham, W.M., Control of vector discrete-event systems I — the base model, *IEEE Transactions on Automatic Control*, 38(8), 1214, 1993.
[13] Li, Y. and Wonham, W.M., Control of vector discrete-event system II. Controller synthesis, *IEEE Transactions on Automatic Control*, 39(3), 512, 1994.
[14] Wonham, W.M., Notes on discrete event system control, System Control Group Electrical & Computer Engineering Dept, University of Toronto, 1999.
[15] Moody, J.O. and Antsaklis, P.J., Petri net supervisors for DES with uncontrollable and unobservable transitions, Technical Report of the ISIS Group at the University of Notre Dame, February 1999.

© 2005 by Chapman & Hall/CRC

Appendix

51A.1 Controllable Transitions

The following is a list of the controllable events shown in the control hierarchies. The transition number is the one shown in the relevant Petri net diagram. The hierarchies are denoted as CC for operational command, SC for collaborative sensing, and WC for network communication. Event descriptions self-explanatory.

Trans.#	Hierarchy	Description
1	CC	Selecting Donation Region
2	CC	Altered Coverage Area
4	CC	Area Unmapped/Send Map Messages
5	CC	New Node Assigned to Network
6	CC	Sending Alterations
7	CC	New Region Priority
9	CC	Initial Mapping Complete
10	CC	Send Deployment Notices
11	CC	Resources Found/Sending Coverage Adjustments
12	CC	Wake Command
13	CC	Receive Cluster Status Message
14	CC	Sleep Command
15	CC	Topology Request from User/Query Clusters
16	CC	Resource Request
17	CC	Recall Command
18	CC	Receives Cluster Statistics
19	CC	Recall Notices Sent
20	CC	Network Statistics to User
21	CC	Sending Message
22	CC	Poll Clusters
23	CC	Response TO, Respond to User
24	CC	Drain Counter
25	CC	Response Received
28	CC	Stop Drain
30	CC	Altered Coverage Area
31	CC	Deployment Command
32	CC	Coverage Commands
33	CC	Wake Command
35	CC	New Node Assigned to Cluster
36	CC	Sending Deployment Notices
38	CC	Cluster Topology Request Root
39	CC	Recall Command
40	CC	Receive Resource Query
41	CC	Receive Cluster Status Request
42	CC	Send Recall Notice
43	CC	Sending Topology Report
44	CC	Sending Message
45	CC	Response TO
46	CC	Poll Leaves
49	CC	Drain Counter
50	CC	Response Received
51	CC	Stop Drain
55	CC	Coverage Area Adjusted
56	CC	New Coordinates Reached
58	CC	Send Map Update
61	CC	Update Requested
63	CC	Statistics Sent

(*Continued*)

© 2005 by Chapman & Hall/CRC

Trans.#	Hierarchy	Description
64	CC	Wake Command
65	CC	Recalled
66	CC	Receive Resource Query
67	CC	Retreat Complete
69	CC	Response to CH
70	CC	Send Message
71	SC	Receive Message from User
72	SC	Receive Message from CH
73	SC	Cluster Boundaries and Paths Message
74	SC	Adjust the Paths Detectable Probability Threshold
75	SC	CH Event Summary
76	SC	Waiting TO, Compute Overall Coverage
77	SC	Leaf Movement for Optimal Cluster Coverage
78	SC	Sensor Movement for Prioritized Region
79	SC	Surveillance Topology Request
81	SC	Gap Coverage Not Found
86	SC	Send Fusion Data Message to User
88	SC	Receive Event Statistics
89	SC	Increase Threshold
91	SC	Send to On Board Sensor Fusion
92	SC	Signal Sensed
96	SC	Receiving Message from Root
97	SC	Move Finished
100	SC	Send to Onboard Fusion
101	SC	Cluster Optimal Coverage Movement
102	SC	Receive Message from Leaf
104	SC	Low Noise
105	SC	Sleep Wake
106	SC	Sensor Movement for Prioritized Relocation
107	SC	Leaf node Location and Characteristics Message
108	SC	Cluster Self Movement
109	SC	NonSelf Movement
110	SC	Movement Finished
111	SC	Adjust Paths/Detect Probability
112	SC	Surveillance Topology Request
113	SC	Waiting TO
114	SC	Waiting TO
115	SC	Retain Leaf Node Status
117	SC	Computing Boundaries Complete
118	SC	Latency TO
119	SC	Send to Onboard Sensor Fusion
120	SC	Receive Event Statistics
121	SC	Threshold Increased
123	SC	Send to Onboard Sensor Fusion
126	SC	Movement Complete
127	SC	Receive Message from CH
130	SC	Prioritized Location Movement
131	SC	Surveillance Topology Request
135	SC	Sleep TO
136	SC	Leaf Node Move Command
137	SC	Low Noise
138	SC	On Board Fusion
139	SC	Movement Complete
140	SC	Location and Characteristics to CH
141	SC	Occlusion Move Complete
142	WC	Receive Message from CH
147	WC	Message Intact

(*Continued*)

© 2005 by Chapman & Hall/CRC

Trans.#	Hierarchy	Description
149	WC	Self Demotion
150	WC	Move Complete
153	WC	Move Complete
154	WC	Receive Message from User
155	WC	Send Message
156	WC	Request Retransmit
157	WC	Move Message
158	WC	Update User
160	WC	Signal Power Message
161	WC	Integrity Adjusted
162	WC	General Message
163	WC	Receive User ACK
165	WC	Frequency Hopping Message
166	WC	FH Adjusted
167	WC	Processing Complete
168	WC	Move Complete
169	WC	SPI Failure
170	WC	Send Retransmit Request
174	WC	Move
179	WC	Receive Message
182	WC	Demotion TO
184	WC	Send Hello to Root
185	WC	Message Intact
190	WC	Request Retransmit
191	WC	Move Complete
192	WC	Receive Message from Root
193	WC	Move Command
194	WC	Update Root
195	WC	Receive Root ACK
196	WC	Signal Power Message
197	WC	Integrity Adjusted
198	WC	ACK TO
199	WC	Frequency Hopping Command
200	WC	FH Complete
201	WC	General Message
202	WC	Move Complete
204	WC	Retain SH Status
205	WC	Self Demotion
206	WC	Processing Complete
207	WC	SPI Failure
208	WC	Send Retransmit
210	WC	Wake Message
211	WC	Send Hello
213	WC	Move Command
214	WC	Move Complete
215	WC	Receive Message from CH
216	WC	Event ACK not Received
217	WC	Retain Leaf Status
218	WC	Signal Power Message
219	WC	Adjustment Complete
220	WC	Frequency Hopping Message
221	WC	FH Complete
222	WC	Interpreting Signal Integrity
223	WC	Send Event Summary
225	WC	General Message
226	WC	Send Message
229	WC	Wake Message
230	WC	Processing Complete
231	WC	Move Complete

© 2005 by Chapman & Hall/CRC

51A.2 Uncontrollable Transitions

The following is a list of the uncontrollable events shown in the control hierarchies. The transition number is the one shown in the relevant Petri net diagram. The hierarchies are denoted as CC for operational command, SC for collaborative sensing, and WC for network communication. Event descriptions are self-explanatory.

Trans.#	Hierarchy	Description
3	CC	Destroyed
8	CC	Insufficient Resources for Coverage Alteration
26	CC	Demotion to Cluster Head
27	CC	Promotion to Root
29	CC	Destroyed
34	CC	Sleep
37	CC	Attacked
45	CC	Timeout on Response
47	CC	Promoted to Cluster Head
48	CC	Demoted to Leaf
52	CC	Region Mapping Complete
53	CC	Path Obstructed
54	CC	Destroyed
57	CC	Deployed
59	CC	Low Power
60	CC	Damaged
62	CC	Attacked
68	CC	Sleep
80	SC	Waiting Timeout
82	SC	Promotion
83	SC	Demotion
84	SC	Gap in Coverage
90	SC	Background False Alarm
93	SC	White Noise
94	SC	Interference
95	SC	Spike Noise
98	SC	Occlusion
99	SC	Excessive Unrecoverable Noise
103	SC	Saturation Signal Detected
113	SC	Waiting Timeout
116	SC	Promoted
122	SC	Background False Alarm
124	SC	Signal Alarm
125	SC	White Noise
128	SC	Interference/Jamming
129	SC	Spike Noise
132	SC	Occlusion
133	SC	Excessive Unrecoverable Noise
134	SC	Saturation Signal Detected
143	WC	Corrupt Message
144	SC	Frequency Hopping Message
145	SC	Signal Power Message
146	WC	Position Problem
148	WC	Demoted
159	WC	Re-contact User
171	WC	Corrupt Message
172	WC	Dies
173	WC	Dies
177	WC	Promoted

(*Continued*)

© 2005 by Chapman & Hall/CRC

Trans.#	Hierarchy	Description
178	WC	Overdue Hello
181	WC	Dies
183	WC	Low Power
187	WC	Signal Power Problem
188	WC	Frequency Hopping Problem
189	WC	Position Problem
209	WC	Corrupt Message
203	WC	Sleep Command
212	WC	Promotion
224	WC	Target Event Sensed
227	WC	Dies
228	WC	Sleep
234	WC	Corrupt Message

51A.3 Petri Net Controller Implementation

The controller presented in this section adds new places and transitions to the Petri net plant models to enforce the control specifications. It has been defined using the methodology described in the following list of constraints. Added controller places and arcs are given

51A.3.1 Define Controller Specifications

1. When a node is waiting for on-board data fusion, it should be prevented from moving by WC, CC and SC. Also, it should wait to be promoted by WC or by SC until sensing is complete.

$P57 + P120 < 2$	$P57 + P115 < 2$	$P57 + P107 < 2$	$P57 + P88 < 2$
$P57 + P96 < 2$	$P57 + P39 < 2$	$P57 + P130 < 2$	

2. Sleep state in SC caused by unrecoverable noise or a saturated signal should also force the sleep state in WC and CC, and *vice versa* for the case of a leaf node. Wakeup in SC needs to send wake-up to CC/WC. To enforce the above control specification, we added:

   Inhibitor arc from P76 to all transitions in WC leaf hierarchy
   Inhibitor arc from P76 to all transitions in CC leaf hierarchy

3. Not a conflict issue, requires intra-plant transition to force all hierarchies into a reasonable state. Moving and self-location events cannot co-exist.

$P79 + P120 < 2$	$P79 + P115 < 2$	$P79 + P107 < 2$	$P79 + P88 < 2$
$P79 + P96 < 2$	$P79 + P39 < 2$	$P79 + P130 < 2$	

4. While the node is in the process of dynamically updating the cluster head (receiving all statistics events), it should also be prevented from moving by WC, CC, or SC until a decision is made.

$P47 + P120 < 2$	$P47 + P115 < 2$	$P47 + P107 < 2$	$P47 + P88 < 2$
$P47 + P96 < 2$	$P47 + P39 < 2$	$P47 + P130 < 2$	

© 2005 by Chapman & Hall/CRC

5. While the node is awaiting location and characteristics from a leaf (receiving all statistics events), it should also be prevented from moving by WC, CC, or SC until a decision is made.

P62 + P120 < 2	P62 + P115 < 2	P62 + P107 < 2	P62 + P88 < 2
P62 + P96 < 2	P62 + P39 < 2	P62 + P130 < 2	

6. Sensor in low-power mode, as determined by WC, or damaged mode, as determined by CC, should be prohibited from any movements in SC as a result of prioritized relocation or occlusion adjustments. To enforce the above control specifications, we added:

   Inhibitor arcs from P126 to transitions:

   T130 T132 T136 T72 T53 T57 T58

   Inhibitor arcs from P31 to transitions:

   T130 T132 T136 T232 T213 T193 T207 T157 T169

7. Retreat in CC should supersede all actions in WC/SC, except propagation of retreat command. To enforce the above control specifications, we added:

   Inhibitor arc from P33 to all transitions leaving listening state
   Inhibitor arc from P22 to all transitions leaving listening state
   Arc from retreat signal to retreat place in all hierarchies

8. Entrance into the damaged state in CC should force entrance to the low-power state in WC and *vice versa*. To enforce the above control specifications, we added:

   Arc from T222 to P31
   Arc from T59 to P126
   Arc from T60 to P126

9. Nodes encountering a target signal in the SC should suspend mapping action in CC until sensing is complete.

P70 + P28 < 2

10. Move commands in CC/WC should be delayed while node is receiving sensing statistics from below.

P62 + P120 < 2	P43 + P120 < 2	P62 + P130 < 2	P43 + P130 < 2
P62 + P115 < 2	P43 + P115 < 2	P62 + P107 < 2	P43 + P107 < 2
P62 + P96 < 2	P43 + P96 < 2	P62 + P88 < 2	P43 + P88 < 2
P62 + P39 < 2	P43 + P39 < 2	P62 + P28 < 2	P43 + P28 < 2

51A.3.2 Controller Implementation for Unexplained Control Specifications

To enforce control specification	Added controller places	Arc to transitions	Arc from transitions
P57 + P120 < 2	P137	T100, T213	T114, T214
P57 + P115 < 2	P138	T100, T207	T114, T202
P57 + P107 < 2	P139	T100, T193	T114, T191
P57 + P88 < 2	P140	T100, T158	T114, T153
P57 + P96 < 2	P141	T100, T169	T114, T168
P57 + P39 < 2	P142	T100, T72	T114, T75

(*Continued*)

© 2005 by Chapman & Hall/CRC

To enforce control specification	Added controller places	Arc to transitions	Arc from transitions
$P57 + P130 < 2$	P143	T100, T232	T114, T231
$P79 + P120 < 2$	P147	T131, T213	T140, T214
$P79 + P115 < 2$	P148	T131, T207	T140, T202
$P79 + P107 < 2$	P149	T131, T193	T140, T191
$P79 + P88 < 2$	P150	T131, T157	T140, T153
$P79 + P96 < 2$	P151	T131, T169	T140, T168
$P79 + P39 < 2$	P152	T131, T72	T140, T75
$P79 + P130 < 2$	P153	T131, T232	T140, T231
$P47 + P120 < 2$	P157	T88, T213	T87, T214
$P47 + P115 < 2$	P158	T88, T207	T87, T202
$P47 + P107 < 2$	P159	T88, T193	T87, T191
$P47 + P88 < 2$	P160	T88, T157	T87, T150
$P47 + P96 < 2$	P161	T88, T169	T87, T168
$P47 + P39 < 2$	P162	T88, T72	T87, T75
$P47 + P130 < 2$	P163	T88, T232	T87, T231
$P62 + P120 < 2$	P167	T107, T213	T113, T214
$P62 + P115 < 2$	P168	T107, T207	T113, T202
$P62 + P107 < 2$	P169	T107, T193	T113, T191
$P62 + P88 < 2$	P170	T107, T157	T113, T150
$P62 + P96 < 2$	P171	T107, T169	T113, T168
$P62 + P39 < 2$	P172	T107, T72	T113, T75
$P62 + P130 < 2$	P173	T107, T232	T113, T231
$P70 + P28 < 2$	P190	T124, T58	T122, T52
$P62 + P120 < 2$	P200	T107, T213	T113, T214
$P43 + P120 < 2$	P201	T75, T213	T80, T214
$P62 + P130 < 2$	P202	T107, T193	T113, T191
$P43 + P130 < 2$	P203	T75, T232	T80, T231
$P62 + P115 < 2$	P204	T107, T207	T113, T202
$P43 + P115 < 2$	P205	T75, T207	T80, T202
$P62 + P107 < 2$	P206	T107, T193	T113, T191
$P43 + P107 < 2$	P207	T75, T193	T80, T191
$P62 + P96 < 2$	P208	T107, T169	T113, T168
$P43 + P96 < 2$	P209	T75, T169	T80, T168
$P62 + P88 < 2$	P210	T107, T157	T113, T150
$P43 + P88 < 2$	P211	T75, T157	T80, T150
$P62 + P39 < 2$	P212	T107, T72	T113, T75
$P62 + P28 < 2$	P214	T107, T57, T53, T58	T113, T52
$P43 + P28 < 2$	P215	T75, T57, T53, T58	T80, T52

51A.4 FSM and Vector Controller Implementation

Boolean functions derived as the FSM controller are exerted on those controllable events to prevent violation of the control specifications. The controllable transitions are allowed to fire provided corresponding Boolean functions are satisfied.

The state vector of the system is the concatenation of the state vector of a node in three different hierarchies. It is important to note here that the node roles in the hierarchies are independent. A node occupying the cluster head level in the sensor coverage hierarchy is allowed to occupy any of the three levels in the other two hierarchies and is not restricted in any fashion.

1. For transition 128, it can fire, i.f.f. it is enabled and the state status satisfies the conjunction of the following predicates:

$P75 + P144 = 0$	$P75 + P137 = 0$	$P75 + P120 = 0$	$P75 + P130 = 0$
$P75 + P39 = 0$	$P75 + P145 = 0$	$P75 + P28 = 0$	$P75 + P115 = 0$
$P75 + P107 = 0$	$P75 + P88 = 0$	$P75 + P96 = 0$	

© 2005 by Chapman & Hall/CRC

2. For transition 241, it can fire, i.f.f. it is enabled and the state status satisfies the conjunction of the following predicates:

P75 + P144 = 0	P138 + P144 = 0	P80 + P144 = 0	P126 + P144 = 0

3. For transition 140, it can fire, i.f.f. it is enabled and the state status satisfies the conjunction of the following predicates:

P75 + P137 = 0	P138 + P137 = 0	P80 + P137 = 0	P126 + P137 = 0
P137 + P121 = 0	P137 + P120 = 0	P137 + P125 = 0	P137 + P133 = 0
P137 + P107 = 0	P137 + P111 = 0	P137 + P113 = 0	P137 + P112 = 0
P137 + P88 = 0	P137 + P91 = 0	P137 + P94 = 0	P137 + P98 = 0

4. For transition 213, it can fire, i.f.f. it is enabled and the state status satisfies the conjunction of the following predicates:

P75 + P120 = 0	P138 + P120 = 0	P137 + P120 = 0	P39 + P120 = 0
P54 + P120 = 0	P43 + P120 = 0	P59 + P120 = 0	P63 + P120 = 0
P49 + P120 = 0			

5. For transition 232, it can fire, i.f.f. it is enabled and the state status satisfies the conjunction of the following predicates:

P75 + P130 = 0	P138 + P130 = 0	P80 + P130 = 0	P59 + P130 = 0
P43 + P130 = 0	P63 + P130 = 0	P49 + P130 = 0	

6. For transition 55, it can fire, i.f.f. it is enabled and the state status satisfies the conjunction of the following predicates:

P75 + P39 = 0	P138 + P39 = 0	P80 + P39 = 0	P39 + P120 = 0
P39 + P121 = 0	P39 + P125 = 0	P39 + P133 = 0	P39 + P107 = 0
P39 + P111 = 0	P39 + P113 = 0	P39 + P112 = 0	P39 + P88 = 0
P39 + P91 = 0	P39 + P94 = 0	P39 + P98 = 0	P54 + P39 = 0
P43 + P39 = 0	P63 + P39 = 0	P49 + P39 = 0	P59 + P39 = 0

7. For transition 131, it can fire, i.f.f. it is enabled and the state status satisfies the following predicate:

P75 + P145 = 0

8. For transition 100, it can fire, i.f.f. it is enabled and the state status satisfies the conjunction of the following predicates:

P54 + P70 = 0	P54 + P115 = 0	P54 + P107 = 0	P54 + P146 = 0
P54 + P120 = 0	P54 + P88 = 0	P54 + P96 = 0	P54 + P39 = 0
P54 + P28 = 0			

© 2005 by Chapman & Hall/CRC

9. For transition 117, it can fire, i.f.f. it is enabled and the state status satisfies the conjunction of the following predicates:

P54 + P70 = 0	P59 + P70 = 0	P63 + P70 = 0	P126 + P70 = 0

10. For transition 207, it can fire, i.f.f. it is enabled and the state status satisfies the conjunction of the following predicates. P54 + P115 = 0 P59 + P115 = 0 P63 + P115 = 0 P75 + P115 = 0 P138 + P115 = 0 P80 + P115 = 0 P43 + P115 = 0 P49 + P115 = 0.
11. For transition 193, it can fire, i.f.f. it is enabled and the state status satisfies the conjunction of the following predicates:

P54 + P107 = 0	P59 + P107 = 0	P63 + P107 = 0	P75 + P107 = 0
P138 + P107 = 0	P137 + P107 = 0	P80 + P107 = 0	P43 + P107 = 0
P49 + P107 = 0	P39 + P107 = 0		

12. For transition 91, it can fire, i.f.f. it is enabled and the state status satisfies the following predicate:

P54 + P146 = 0

13. For transition 235, it can fire, i.f.f. it is enabled and the state status satisfies the conjunction of the following predicates:

P138 + P144 = 0	P138 + P137 = 0	P138 + P120 = 0	P138 + P130 = 0
P138 + P39 = 0	P138 + P107 = 0	P138 + P115 = 0	

14. For transition 137, it can fire, i.f.f. it is enabled and the state status satisfies the conjunction of the following predicates:

P80 + P144 = 0	P80 + P137 = 0	P80 + P120 = 0	P80 + P130 = 0
P80 + P39 = 0	P80 + P115 = 0	P80 + P107 = 0	P80 + P96 = 0
P80 + P88 = 0			

15. For transition 96, it can fire, i.f.f. it is enabled and the state status satisfies the conjunction of the following predicates:

P59 + P70 = 0	P59 + P115 = 0	P59 + P107 = 0	P59 + P130 = 0
P59 + P120 = 0	P59 + P88 = 0	P59 + P96 = 0	P59 + P39 = 0

16. For transition 103, it can fire, i.f.f. it is enabled and the state status satisfies the conjunction of the following predicates:

P63 + P70 = 0	P63 + P115 = 0	P63 + P107 = 0	P63 + P130 = 0
P63 + P88 = 0	P63 + P120 = 0	P63 + P96 = 0	P63 + P39 = 0

17. For transition 222, it can fire, i.f.f. it is enabled and the state status satisfies the conjunction of the following predicates:

P126 + P144 = 0	P126 + P137 = 0	P126 + P70 = 0

© 2005 by Chapman & Hall/CRC

18. For transition 53, it can fire, i.f.f. it is enabled and the state status satisfies the conjunction of the following predicates:

$P75 + P28 = 0$	$P54 + P28 = 0$

19. For transition 57, it can fire, i.f.f. it is enabled and the state status satisfies the conjunction of the following predicates:

$P75 + P28 = 0$	$P54 + P28 = 0$

20. For transition 218, it can fire, i.f.f. it is enabled and the state status satisfies the conjunction of the following predicates:

$P137 + P121 = 0$	$P39 + P121 = 0$

21. For transition 220, it can fire, i.f.f. it is enabled and the state status satisfies the conjunction of the following predicates:

$P137 + P125 = 0$	$P39 + P125 = 0$

22. For transition 234, it can fire, i.f.f. it is enabled and the state status satisfies the conjunction of the following predicates:

$P137 + P133 = 0$	$P39 + P133 = 0$

23. For transition 74, it can fire, i.f.f. it is enabled and the state status satisfies the conjunction of the following predicates:

$P43 + P88 = 0$	$P43 + P96 = 0$	$P43 + P130 = 0$	$P43 + P120 = 0$
$P43 + P115 = 0$	$P43 + P107 = 0$	$P43 + P96 = 0$	$P43 + P88 = 0$
$P43 + P39 = 0$			

24. For transition 157, it can fire, i.f.f. it is enabled and the state status satisfies the conjunction of the following predicates:

$P43 + P88 = 0$	$P75 + P88 = 0$	$P54 + P88 = 0$	$P80 + P88 = 0$
$P59 + P88 = 0$	$P43 + P88 = 0$	$P63 + P88 = 0$	$P49 + P88 = 0$
$P137 + P88 = 0$	$P39 + P88 = 0$		

25. For transition 169, it can fire, i.f.f. it is enabled and the state status satisfies the conjunction of the following predicates:

$P43 + P96 = 0$	$P75 + P96 = 0$	$P54 + P96 = 0$	$P80 + P96 = 0$
$P59 + P96 = 0$	$P43 + P96 = 0$	$P63 + P96 = 0$	$P49 + P96 = 0$

© 2005 by Chapman & Hall/CRC

26. For transition 84, it can fire, i.f.f. it is enabled and the state status satisfies the conjunction of the following predicates:

$P49 + P130 = 0$	$P49 + P120 = 0$	$P49 + P115 = 0$	$P49 + P107 = 0$
$P49 + P96 = 0$	$P49 + P88 = 0$	$P49 + P39 = 0$	

27. For transition 196, it can fire, i.f.f. it is enabled and the state status satisfies the conjunction of the following predicates:

$P137 + P111 = 0$	$P39 + P111 = 0$

28. For transition 199, it can fire, i.f.f. it is enabled and the state status satisfies the conjunction of the following predicates:

$P137 + P113 = 0$	$P39 + P113 = 0$

29. For transition 209, it can fire, i.f.f. it is enabled and the state status satisfies the conjunction of the following predicates:

$P137 + P112 = 0$	$P39 + P112 = 0$

30. For transition 160, it can fire, i.f.f. it is enabled and the state status satisfies the conjunction of the following predicates:

$P137 + P91 = 0$	$P39 + P91 = 0$

31. For transition 165, it can fire, i.f.f. it is enabled and the state status satisfies the conjunction of the following predicates:

$P137 + P94 = 0$	$P39 + P94 = 0$

32. For transition 171, it can fire, i.f.f. it is enabled and the state status satisfies the conjunction of the following predicates:

$P137 + P98 = 0$	$P39 + P98 = 0$

© 2005 by Chapman & Hall/CRC

51A.5 Surveillance Network Petri Nets Plant Models

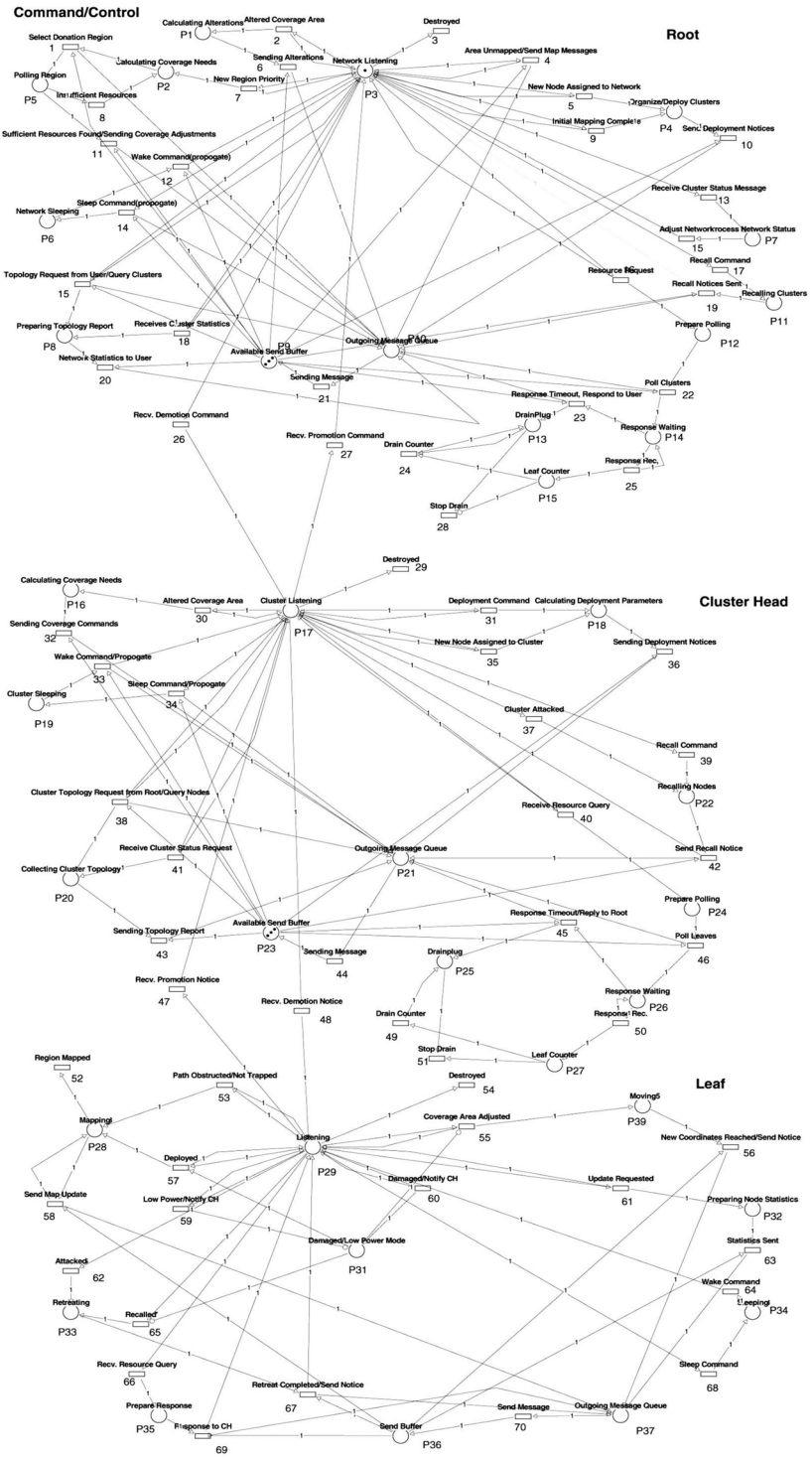


Figure 51A.1. Operational command hierarchy.

© 2005 by Chapman & Hall/CRC

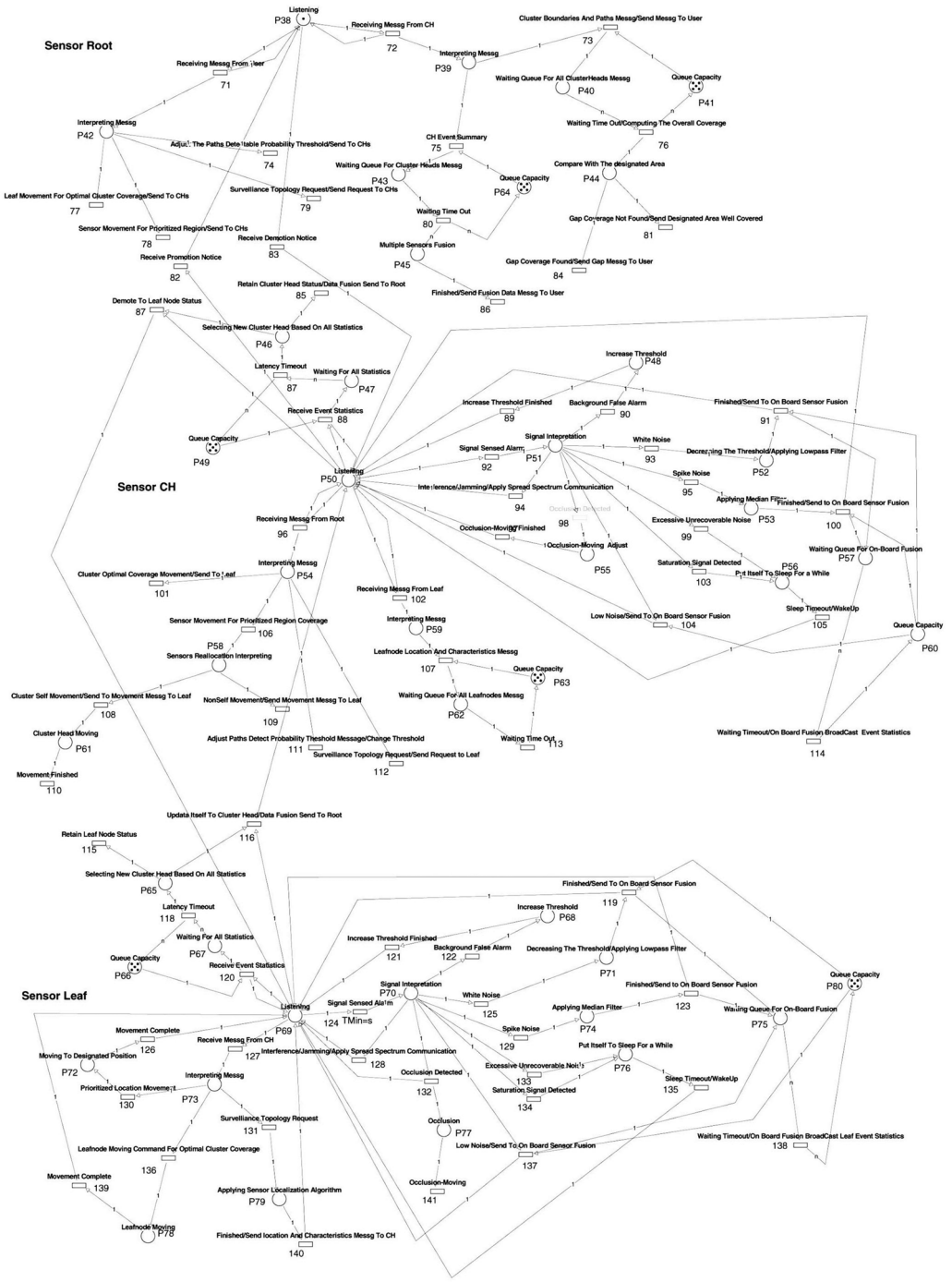


Figure 51A.2. Collaborative sensing hierarchy.

© 2005 by Chapman & Hall/CRC

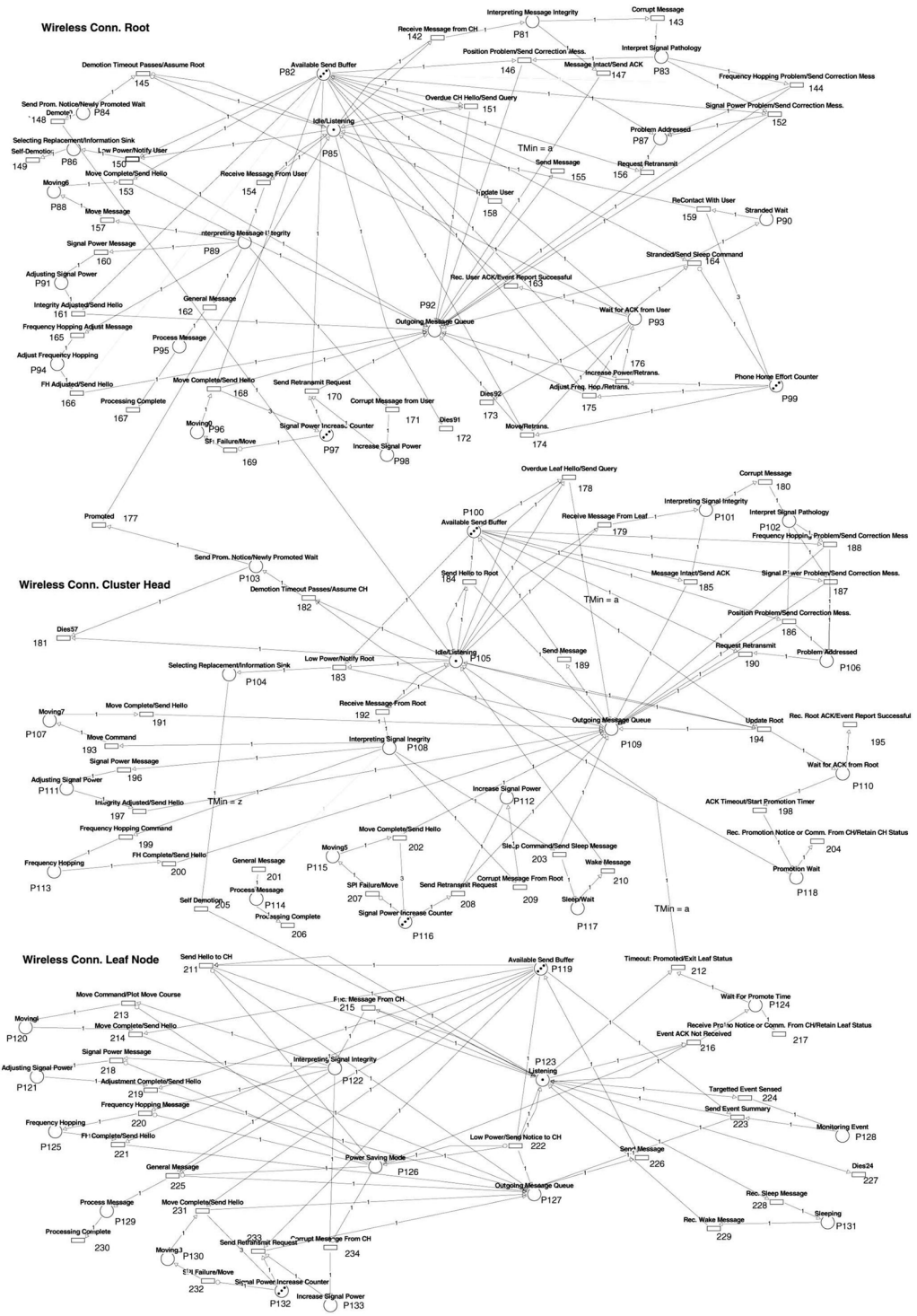


Figure 51A.3. Network communication hierarchy.

© 2005 by Chapman & Hall/CRC

IX

Engineering Examples

52. SenSoft: Development of a Collaborative Sensor Network *Gail Mitchell, Jeff Mazurek, Ken Theriault, and Prakash Manghwani* *1011*
Introduction • Overview of SensIT System Architecture • Prototype Hardware Platform • SenSoft Architectural Framework • Software Infrastructure • SenSoft Signal Processing • Component Interaction • An Example • Summary

53. Statistical Approaches to Cleaning Sensor Data *Eiman Elnahrawy and Badri Nath* *1023*
Introduction • Bayesian Estimation and Noisy Sensors • Error Models and Priors • Reducing the Uncertainty • Traditional Query Evaluation and Noisy Sensors • Querying Noisy Sensors • Spatio-Temporal Dependencies and Wireless Sensors • Modeling and Dependencies • Online Distributed Learning • Detecting Outliers and Recovery of Missing Values • Future Research Directions

54. Plant Monitoring with Special Reference to Endangered Species *K.W. Bridges and Edo Biagioni* *1039*
Introduction • The Monitoring System • Typical Studies Involving Plant Monitoring • Sensor Networks • Data Characteristics and Sensor Requirements • Spatial and Temporal Scales: Different Monitoring Requirements • Network Characteristics • Deployment Issues • Data Utilization

55. Designing Distributed Sensor Applications for Wireless Mesh Networks *Robert Poor and Cliff Bowman* *1049*
Introduction • Characteristics of Mesh Networking Technology • Comparison of Popular Network Topologies • Basic Guidelines for Designing Practical Mesh Networks • Examples of Practical Mesh Network Applications

© 2005 by Chapman & Hall/CRC

Sensor networks have become an important source of information with numerous real life applications. Sensor networks are sued for monitoring transportation and traffic control, contamination level in soil and water, climate, building structure, habitat, quality of perishable food items, etc. In this section, the emphasis is on monitoring of important components within the environment and to clean the date before decision-making.

Mitchell *et al.* describe a filed test performed as a part of the DARPA-IXO initiative. It took place at a military installation. The SITEX test was the first time that many of the technologies described in this book were used in a realistic military situation. The architecture, its components, and results of the test are described in detail.

Elnahrawy and Badrinath emphasize on online cleaning of the sensor data before any crucial decisions are taken. Data collected from wireless sensor networks (WSNs) are subject to several problems and source of errors. These problems may seriously impact the actual usage of such networks and yield imprecise and inaccurate answers for any query on sensor data. The authors focus on probabilistic, efficient, and scalable approaches in reducing the effect of random errors, Bayesian estimation, traditional query evaluation, querying noisy sensors, online distributed learning, spatio-temporal dependencies and wireless sensors, detection of outliers and malicious sensors and recovery of missing values.

Bridges and Biagioni emphasize on monitoring the phenology of endangered species of plants and their surrounding environment by sensor networks. The weather information that is collected for phonological studies includes the air temperature, rainfall amount, relative humidity, solar radiation intensity, and wind speed and direction. They discuss monitoring devices like digital sensor, thermo couple sensor, tipping bucket sensor, digital camera, and networking and deployment of sensors and data utilization.

Poor *et al.* have experience in the implementation and use of sensor networks for industrial applications. Their chapter provides application design rules. These rules are based on experience. They show how these design principles can be used to implement systems using off-the-shelf components.

In summary, the section looks at specific sensor network implementations and lessons learned by fielding the systems.

© 2005 by Chapman & Hall/CRC

52

SenSoft: Development of a Collaborative Sensor Network

Gail Mitchell, Jeff Mazurek, Ken Theriault, and Prakash Manghwani

52.1 Introduction

In 1999, the Defense Advanced Research Projects Agency (DARPA) established the Sensor Information Technology (SensIT) program to investigate the feasibility of employing thousands of autonomous, distributed, networked, multi-modal ground sensors to accomplish intelligence, surveillance and reconnaissance tasks. A large group of Principal Investigators was challenged to develop innovative hardware, algorithms, and software to demonstrate the potential of distributed micro-sensor networks. These investigators collaborated with BBN Technologies, as the system architect and integrator, to develop and demonstrate a system of distributed, networked sensor nodes with in-network data processing, target detection, classification, and tracking, and communication within and outside of the network. The SensIT architecture and SensIT Software system (SenSoft) described in this chapter are the results of this collaboration. This prototype distributed sensor network, and the research accomplished by the contributors in the process of achieving the prototype, represents a firm foundation for further development of, and experimentation with, information technology for distributed sensor systems.

52.2 Overview of SensIT System Architecture

A SensIT network is a system of sensor nodes communicating with each other within the network and also communicating with entities outside the SensIT network. The conceptual architecture for such a system is depicted in Figure 52.1. As shown here, a SensIT network is a robust, flexible collection of "smart" sensor nodes; nodes that include, in addition to sensing capabilities, processing and communications functionality that enable building *ad hoc*, flexible, and robust networks for distributed signal processing. For example, the network of nodes shown in Figure 52.1 might be deployed in hostile territory and receiving commands (tasks) from a mobile base station (the trucks). Similarly, a soldier in a remote platoon uses a handheld device to request information about activity sensed by the network,

© 2005 by Chapman & Hall/CRC

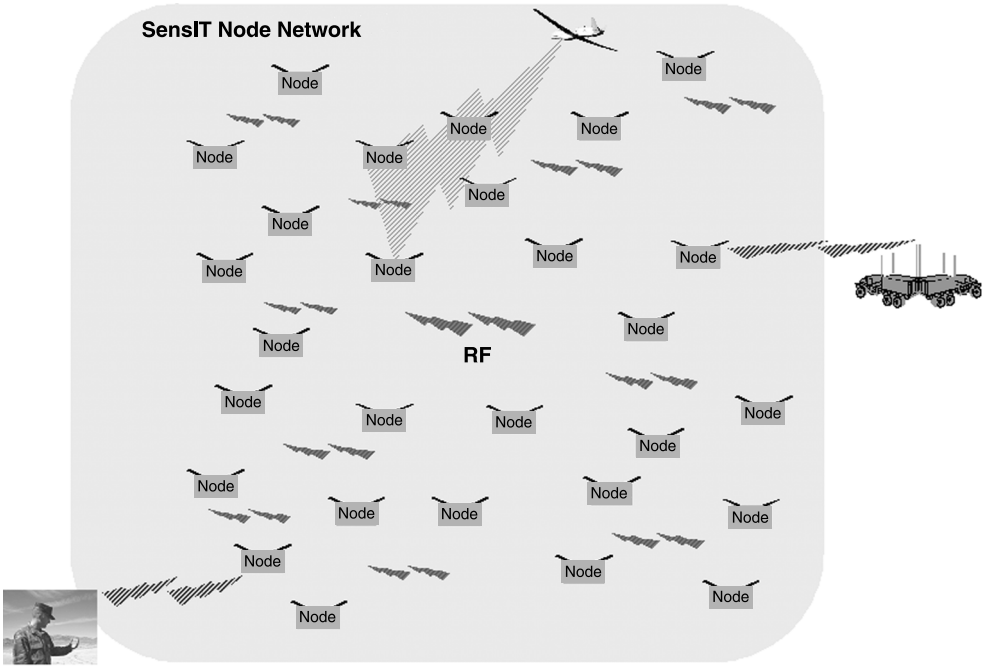


Figure 52.1. SensIT system architecture concept.

and an unmanned aerial vehicle sends commands to request information that it relays to a ship waiting offshore. In each example a base station, i.e. a user of the network, connects to a node in the network to give commands to, or receive information from, the network as a whole. The node with which a base station communicates is, at that point in time, the network's gateway node. The network gateway may or may not be the same node for all extra-network communications; the gateway node used by a base station site might even change if that site is mobile or if connectivity changes.

The key to making this happen is that data are obtained and commands are processed within the node network. Unlike distributed sensor systems that collect sensing information and move it to a centralized location for processing, the nodes in a SensIT network communicate with each other and cooperate to process the data themselves to accomplish tasks such as processing sensor signals and determining the presence and activity patterns of a target. As an excellent example of a centrally processed distributed sensor system, consider the fields of aircraft-deployed sonobuoys used by naval forces to detect, classify, and track submarines acoustically. Although each individual sonobuoy may employ sophisticated in-buoy signal processing, there is no inter-buoy communication and all results are sent to a monitoring aircraft for integration and display. In SensIT, individual sensor nodes employ sophisticated local processing, but they also communicate local results with other nodes and then cooperate with those nodes to process the results further to detect, classify, and track targets. Through node cooperation and interaction, the thesis is that the total amount of processing needed and the amount of data moved through the network or from the network to a display station may be reduced and, indeed, higher quality information may be obtained.

Commands to a SensIT network arise from within the network or from external sources, and are distributed and processed within the network. Commands from external sources are moved into the network through gateway nodes. The tasks needed to execute a command, and the nodes assigned to execute those tasks, are determined within the network. Similarly, requests from external agents for information obtained through in-network processing are processed within the network in such a way that the results are made available through the gateway. SenSoft (SensIT Software) is the name we give to the software applications and services that effect such in-network signal processing.

© 2005 by Chapman & Hall/CRC

52.3 Prototype Hardware Platform

SenSoft was developed for the Sensoria Corporation's WINS NG 2.0 sensor nodes [1]. These nodes prototype the processing power and other hardware functionalities that we expect to see in micro-nodes anticipated for the future, and thus are a good platform for experimentation with the software and operational concepts for distributed micro-sensor networks. Each WINS NG 2.0 node provides the flexibility to experiment with as many as four separate sensor inputs and to communicate with other nodes in the network. Each node also has embedded processing capability built on Linux operating system software, with a wide range of common utilities and applications (e.g. telnet, ftp, vi) available.

A WINS NG 2.0 node has an onboard digital signal processor (DSP), a Linux processor (Hitachi SH4), RAM and flash memory, a global positioning system (GPS) receiver, and two embedded radio-frequency (RF) modems. A key feature supporting sensor signal experimentation is four analog sensor channels that can receive input from various kinds of sensor. These channels are read and processed independently by the DSP. These nodes provide the signal capture, network communications, and computer processing capabilities upon which a distributed sensor network can be built. As an experimentation platform, they also provide the flexibility to test various network solutions. For example, a sensor network can run as an RF communicating network, but the nodes can also use Ethernet for experimentation and debugging. The various communications modes are also valuable for experimentation with the capabilities and performance of Ethernet solutions versus radio communications. And the generic sensor channels allow experimentation with a variety of types and combinations of sensing modality.

The nodes also have a variety of external interfaces to support these capabilities, including:

- Two PCMCIA slots for PCMCIA and CardBus expansion. We have used a slot, for example, to provide wireless Ethernet communications between nodes.
- A serial port for a console interface to the platform (useful for hardware debugging); an Ethernet port, allowing a node to connect to a local-area network.
- Two antenna connectors for the embedded RF modems and an antenna connector for the GPS.

The two modems in each node are used to build network-wide RF connectivity using short-range local radio connections (see Figure 52.4). A Sensoria network is the union of many small local-area networks, each talking at a different RF (the networks use frequency hopping to prevent collision). Each node belongs to two different local networks and can pass messages between those networks; as a result, messages move within the larger network by "hopping" across the different local networks.

52.4 SenSoft Architectural Framework

The SenSoft framework describes how a network of nodes can be used to develop and experiment with algorithms and software for networked sensor processing. The architecture described in this chapter has two major types of component: *infrastructure* components and *signal-processing* components. The *infrastructure* provides common functionality that enables the activities of collaborative *signal-processing* applications in a network of SensIT nodes. Most of this functionality is present on all sensor nodes; some is only on nodes that can act as gateways; and some of the functionality is present on a command and control (C2) system that interacts with a SensIT network via a gateway.

The on-node software architecture is illustrated in Figure 52.2. In this figure, the heavily outlined boxes indicate the functional components of the support infrastructure and the more lightly outlined boxes are signal-processing components. Every node in a SensIT network has the infrastructure components indicated in the figure; the implementations of these components will be identical on all nodes. Similarly, each node may have signal-processing (and possibly other) applications that use the infrastructure components' interfaces and data structures. Typically (or at least in the implementations and experimentation we have seen thus far) the signal-processing applications are also implemented in

© 2005 by Chapman & Hall/CRC

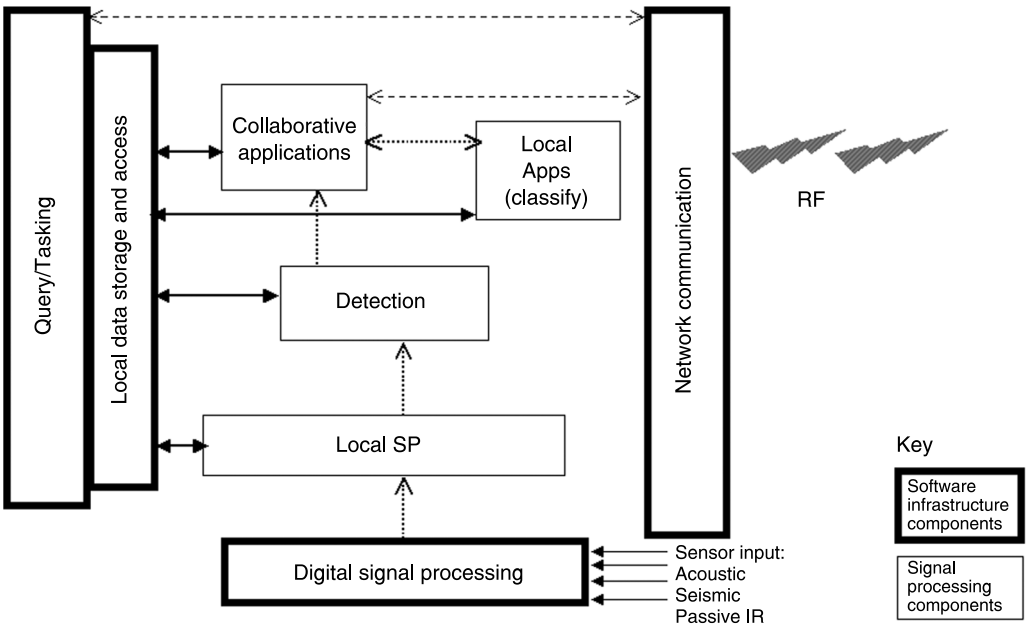


Figure 52.2. SenSoft on-node software architecture.

the same way on all nodes — placing the distributed aspects of the signal-processing computation into the node software.

The architecture for command and control interaction with a SensIT network is illustrated in Figure 52.3. Note that this interaction combines on-node and off-node software (and hardware) components: one or more sensor nodes can be outfitted with task management software that allows them to interact with stations outside the node network. A user of the network (i.e. a C2 system) has a local version of the task management interface; and the C2 station has some sort of communication connectivity with a gateway node (we usually used Internet protocol [IP] over wireless Ethernet, although this is not a requirement).

52.5 Software Infrastructure

Let us examine the SenSoft infrastructure in a little more detail. On-node components of the infrastructure are illustrated in Figure 52.2; C2 components are needed both on- and off-node, and are illustrated in Figure 52.3. These infrastructure components perform common services for signal-processing applications, thus reducing redundancy and facilitating communications within and between nodes. In addition, interfaces defined to each component are intended to facilitate replacement of any given component with a functionally equivalent component. The on-node components include:

- *Hardware/firmware for digitizing the sensor signal inputs*. The Sensoria DSP on each sensor node can sample at a rate of up to 20 kHz per channel on each of the four separate analog input channels (in the WINS NG 2.0 nodes, all channels had to sample at the same rate). The DSP subsystem builds "sections" of 256 16-bit samples with identifiers and timestamps (and other useful information), and transparently moves the data in fixed-size blocks to circular, first in–first out buffers in the SH-4 host system memory. Signal-processing applications must access the time series data in these buffers in a timely fashion; timestamps and sequencing identifiers can be used to tell whether a block of samples has been overwritten in a buffer. A sampling application

© 2005 by Chapman & Hall/CRC

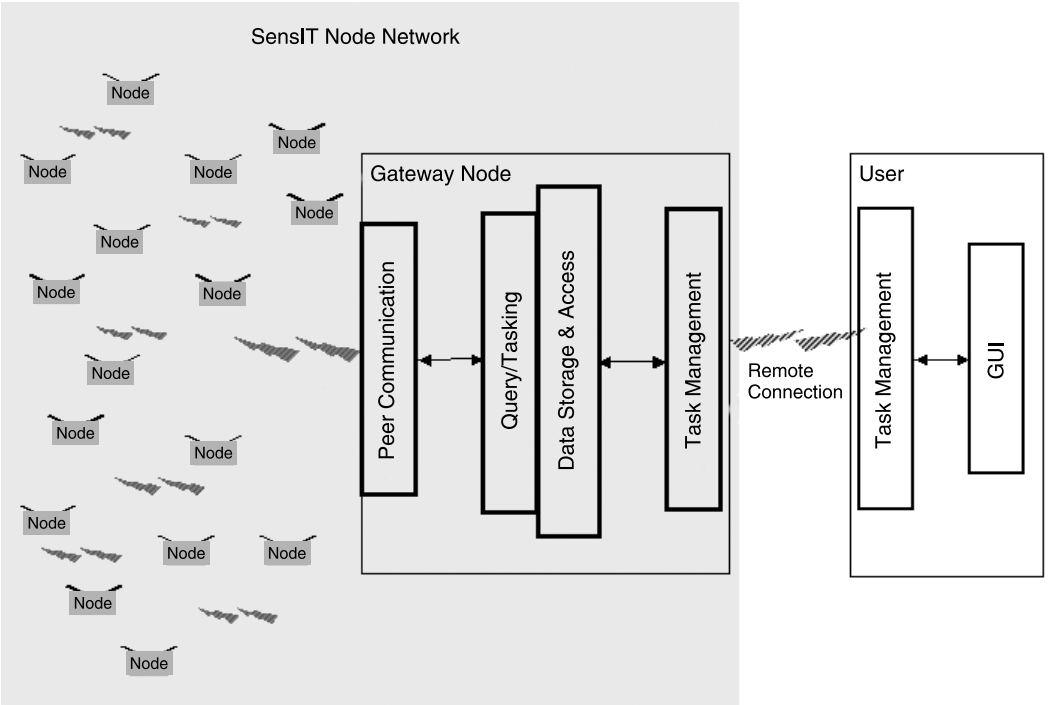


Figure 52.3. SenSoft command and control architecture.

programming interface (API) lets developers select different data rates, select a gain for each sampling channel, and start and stop sampling on each channel. Sensoria documentation [1] provides more detailed information about the sampling API and its use.

- *Network routing for application-level communications between nodes*. Network communications software supports application-level data sharing between nodes and insulates the applications on a node from the mechanics of data transport. In SenSoft, data routing exposes a declarative interface to applications, allowing them to specify what they want to send or receive; the network communications software then determines which messages need to be sent where (routing) and moves the messages over the particular transport mechanism. In SenSoft, data transport was typically over the Sensoria radios, although the software can manage, and we also experimented with, IP-based transport (wired and wireless).
- *Local data storage for data persistence and communications within a node*. In addition to moving data between nodes, it is also necessary to store some data locally on each node for various lengths of time. This data is typically specific to the node — either describing the node, or obtained and analyzed at the node — and can have varying lifetimes that need to be managed by a data storage and access component. For example, the DSP buffers are a transient form of storage for time-stamped digitized signal data. Some of this data, and local analyses of this data, may need to be stored for longer periods than possible in the memory buffer (e.g. until no longer needed by the applications) and, thus, are more appropriately stored and managed by a local "database" system. (Note that a goal of the signal-processing software is to reduce the amount of data that needs to be stored and transported.)

  Data requiring a longer lifetime includes data collected by a node that may be shared at a later point with other nodes, and data (typically, results) that will be moved and processed towards a gateway node for dissemination. For example, target-detection event records may be computed and stored locally, and later sent to neighbors either to trigger actions on their part or in response

© 2005 by Chapman & Hall/CRC

to requests. A data storage and access component mitigates some of the complications of dealing with time delays in data acquisition across nodes.

Data about the state of a particular node or the state of the network itself can also be stored and managed at each node. Examples of this type of data include node configuration and status information (e.g. name, node location, sensor channel settings), codebook values (assigning identifiers to targets, or names to values for display), or application-specific records (such as track updates). In these situations, the aggregate of local storage can be thought of as a database for the network, i.e. the local data storage and access components are pieces of a distributed data management system for the SensIT network.

- *Query/tasking to control movement of queries or tasks into/from nodes, and optionally to perform higher level query processing (e.g. aggregation)*. A query is a request for information; a task is a specification of action(s) that will generate data or information and, thus, can be thought of as defining how to answer a query. Given a request for information from outside the network, the query/tasking component is responsible for determining what processing needs to be done at which nodes in the network to provide the information. For example, a query for current activity within a network might involve all nodes reporting to their neighbors about target detections, and aggregation at certain nodes to provide an overview of amount of activity in different geographic areas of the network. Which nodes perform aggregation might be determined dynamically by the query/tasking component based on dynamic factors such as data availability. Similarly, requests for information that arise within the network and require information from multiple nodes would be managed by the query/tasking components of the nodes.
- *Task management at gateway node(s)*. Each node that can be a gateway for the sensor network must have task management software that can interact with the related task management software at a C2 user station. A gateway node's software must be able to accept tasks (commands), translate those for distribution into the network (through query/tasking), collect results computed within the network, and format and transfer results to the correct C2 user.

A gateway node is one component of the command and control of any system that includes a user of sensor information and a collaborative sensor network (or multiple users, or multiple networks). Although we represent the user as a human, and thus assume the need for a graphical user interface (GUI) for human–machine communications as illustrated in Figure 52.3, a collaborative sensor network will more likely be interacting within a larger operational system and communicating with other computer applications that command the network and use the sensor information obtained. Whether the user component is human or machine, it must include a task/query management component to mediate between the corresponding component at a gateway node and the user application. The user task management component works closely with task management on a gateway node, collectively forming a conduit for information and control flow between the sensor network and the user application (GUI or other).

In all SenSoft experiments, the user interface was a graphical interface for human use. A GUI application provides the ability to send tasks or queries to a sensor network and to display the results of those activities. Display of results creates some interesting issues. For example, should track updates be displayed as soon as they become available at the display station, or should they be displayed in the time sequence in which the events they describe occurred? How do I deal with a result (e.g. track update) that "appears" at the display minutes (or hours) after the event? Time sequencing of results across a large network assumes highly reliable and fast network communications.

52.6 SenSoft Signal Processing

The infrastructure architecture described above is intended to be very general and could be used to support a variety of network applications. The tactical goal (i.e. challenge problem) established for the SensIT community was to support surveillance types of operation with a system for target detection, localization, tracking, and classification. In SenSoft, processing to accomplish these signal-processing

© 2005 by Chapman & Hall/CRC

tasks is done both locally, at individual nodes, and collaboratively within a network of nodes. The components that make up signal processing include:

- *Local signal processing*. Local signal processing is performed on a single node, with multi-modal sensor signal data from only that node. This includes such processing as gain normalization (amplification), filtering (low/high-pass), downsampling, windowing/fast fourier transform, and often also includes threshold detection.
- *Target detection*. Target detection is accomplished through an analysis of sensor signals that determines some anomaly indicating the presence of a target. Target detection will provide the location of the detection (i.e. the location of the node or sensor making the detection) and, depending on the sensing mode, may also locate the target.
- *Collaborative signal processing*. For most sensing modes, target localization and tracking are accomplished through higher level, collaborative signal processing. Nodes with knowledge of local events collaborate by sharing that information with other nodes and cooperating with those nodes to process the shared information to produce target locations and tracks.[1]
- *Target classification*. Classification is the identification of significant characteristics of a target from the signals received. These characteristics could include, for example, method of locomotion (tracked/wheeled) or relative weight (light/medium/heavy) or could more specifically identify the target type (e.g. HMMWV) based on vehicle signature. Classification is typically accomplished locally, due to the large amount of data used. However, classification of a target could also be a cooperative effort (e.g. iterative processing through a sequence of nodes), and a classification component may also collaborate with other signal-processing components. For example, classification could provide information to link a series of tracks as the movement of the same target (i.e. track continuation).

The signal-processing components use the various infrastructure components to pass data and control among themselves on a single node and, for collaborative processing, between nodes. The components also interact with each other in semantically meaningful ways. For example, different algorithms for collaborative signal processing can require different types of target detection (e.g. closest point of approach [CPA] versus signal threshold).

52.7 Component Interaction

The on-node infrastructure and signal-processing components, shown in Figure 52.2 and described above, coordinate (approximately) as follows:

- The DSP moves time series data into on-node buffers where they can be retrieved by other components. This process is defined by the hardware, and described in the WINS 2.0 User's Guide [1].
- A local signal-processing application reads and processes the time series sensor signal data from the DSP buffers. Note that each DSP buffer is circular, so an application must be sensitive to read timing to avoid losing data.
- Results of local signal processing are stored in program buffers or cache (local data storage and access) where they can be retrieved by or sent to other local or remote components. Local signal-processing applications may include a detection component, as noted earlier. Events detected are also stored in cache and shared with other nodes.
- As needed, collaborative applications obtain the results of digital signal processing, local signal processing, and/or detection processing from a group of nodes. The results are obtained as a consequence of queries posed by the application, or through tasks assigned to the application. These results can be processed locally, but the information and control can also be shared with

[1]A track is a location with a speed/direction vector attached.

© 2005 by Chapman & Hall/CRC

collaborators on other nodes via the communication component. Results of such collaborations can provide target, localization, tracks, and classification.
- Collaboration results are stored locally and also passed to other nodes and applications, via the communication mechanisms, for further collaborative processing. For example, a track computed on node A collaborating with nodes B, C, and D could be (temporarily) stored at A and passed to other nodes as appropriate (e.g. in the direction of a target vehicle).

C2 interactions between the sensor network and the user (as shown in Figure 52.3) can coordinate as follows:

- The task manager gathers user-level tasks and queries via the GUI; the task manager may perform global optimization then instructs the query processor on the gateway node as to which tasks/queries are required of the network.
- The query processor optimizes where possible, and moves tasks into nodes for execution (Figure 52.2, described above)
- Task execution provides results via a local query processor; query processing may combine/aggregate the results at a higher level, and moves results to the gateway cache.
- Task manager reads results at the gateway and passes them on to the user.

52.8 An Example

The software infrastructure described in this chapter was designed and implemented in parallel with the research efforts of the SensIT participants; it has been implemented in a variety of ways to incorporate the (intermediate) results of many of those efforts, and at the same time provides a platform for research and experimentation by those participants. In this section we describe the implementation of one collection of infrastructure and signal-processing software named SenSoft v1.

This system was demonstrated in the fall of 2002 on a network of more than 20 sensor nodes placed around a road, as illustrated in Figure 52.4. In the figure, each sensor node is labeled with an integer name (1–27) and with a pair of radio assignments defining intra-network RF radio connectivity. The sensor network was connected from a gateway node (usually node 12) via a wireless Ethernet bridge to a command site (small network of PCs) located inside the building (X marks the approximate spot).

The two radio assignments given for each node are labeled as integer names for local-area networks. Each local network is a star with a base radio (underlined label) and one or more remote radios; a base controls a time-division multiplexed cycle of messages with its remotes and each radio talks only within its network. For example, network 11 is based at node 11 (first radio) and consists of one radio at each of nodes 8, 11, 13 and 23. In Figure 52.4 there are 27 nodes, and thus 54 radios collected into 17 local networks; each arrow points from a remote radio at a node to its corresponding base radio. As noted earlier, messages move through the network by hopping across the local networks. So, for example, a message might move from node 10 to node 16 by following network 12 to node 12, where it hops to network 15 and goes to node 15, where it then hops to network 16 and moves to node 16. Alternately, the same message might follow network 23 through node 23 to node 21, where it hops to network 17, etc.

The network communications component determines the routing that is needed for a message to move through the network. The specific network routing method used by most signal-processing applications in SenSoft is ISI's subject-based routing using SCADDS data diffusion (Scalable Coordination Architectures for Deeply Distributed Systems [2]). This software provides a publish/subscribe API based on attributes describing data of interest; data is exchanged when a publisher sends data for which there are subscriptions and publications with matching attributes. Diffusion uses data attributes to identify *gradients*, or best paths, periodically from publishers to subscribers; gradient information is then used to determine how to route data messages. The efficiency of diffusion can be improved when geographic information is available — essentially, messages can be moved towards

© 2005 by Chapman & Hall/CRC

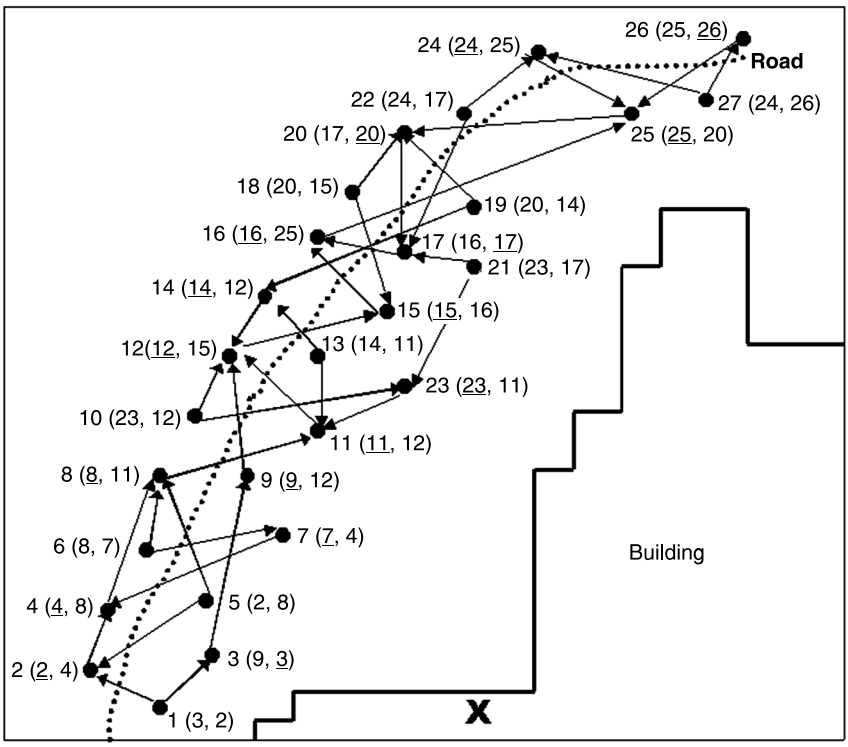


Figure 52.4. SenSoft experimentation network layout.

nodes that are geographically "in a line" with the target node. More detailed (and accurate) information about SCADDS can be obtained elsewhere in this book, and in ISI's SensIT documentation [3].

Local data storage and access is provided in SenSoft v1 in two ways: BAE repositories and Fantastic data distributed cache (FDDC). The BAE repositories are memory-mapped files primarily used for access to recently stored data. Repositories are defined and filled in processes that produce data; a subscription manager allows users in different processes to subscribe to the various repositories, maintains an event queue for each subscription, and notifies subscribers when new data (a repository event) are available. In SenSoft v1, repositories provide access to processed time series data (i.e., data is moved from buffers, processed, and shared via repository) and CPA events for classification and collaborative tracking.

In SenSoft v1, Fantastic cache provides longer term storage of data, and storage of data that will be provided to a user through the gateway. FDDC is a distributed database spread over the node network — each node has a local cache; the collection of nodes forms a database system managed by the FDDC. In SenSoft v1, each local cache provides storage for and access to local node information (e.g. node name, location, sensor configurations, radio configurations), CPA event records (moved from BAE repository), and track update records.

FDDC is implemented as a server process that operates independently and asynchronously from the application programs. It presents an SQL-based interface to application programs, with limited versions of SQL statements such as CREATE TABLE, INSERT, UPDATE, DELETE, and SELECT, and additional statements such as WATCH, PUT, and UNDELETE that are particularly useful in the sensor network environment. It should also be noted that FDDC implicitly handles network routing of data (thus duplicating, to some extent, functionality provided by ISI). In particular, some queries posed against cache can result in the movement of data between nodes to respond to the query or to provide data redundancy for recoverability.

© 2005 by Chapman & Hall/CRC

The SenSoft v1 system demonstrated did not include an in-network query/tasking application (although such an application was demonstrated in other implementations of SenSoft). As a result, signal-processing application processes were always running within the sensor network, producing events and track updates which were all moved to a gateway for transmission to a user. In essence, the task "collect all information" is implicit in the execution of the SenSoft v1.

In SenSoft v1, the same implementations of local signal processing, target detection, tracking, and target classification are run at all nodes. Three sensor channels were used in most testing; one with a microphone, one with a seismometer, and the third with a passive infrared (PIR) detector.

BAE-Austin provides *local signal processing* of the acoustic, PIR, and seismic signals. Time series data produced through the DSP are accessed from the buffer by BAE and processed according to the particular signal type. As noted earlier, sensor input on the WINS NG 2.0 nodes is restricted to be the same frequency for all channels, so BAE also downsamples as needed to achieve appropriate sampling rates for the sensor type. The processed signal data are stored in a BAE repository, along with sampling information. Time tags and gain settings for each data packet are stored in such a way that they are associated with their data block.

In SenSoft v1, BAE also provides target *detection* functionality. Local signal processing detects a target based on the strength of the received signature (e.g. its acoustic signal). Based on the time of occurrence of maximum signal strength (intensity), the detector also provides an estimate of the time of CPA, when the target is closest to the sensor. The sequence of detection outputs is stored in a BAE detection/classification repository; each output record contains detection levels, identification (codebook) values, and other information useful for tracking, data association, and collaborative signal processing.

As noted earlier, in SenSoft v1 the event detections are also stored in Fantastic cache to provide database-style persistence and access to the detection information. To maintain backward compatibility, cache storage of events replicates repository storage; a simple process runs on each node to read detection information from the BAE repository and store it in a cache table on the node.

In SenSoft v1, Penn State University Applied Research Lab (PSU/ARL) [4] provides *collaborative signal processing*, in which nodes collaborate to determine the track of a target. A PSU process subscribes to detection events from its local BAE repository and, via diffusion, can also subscribe to neighboring detections. In the presence of detection events a node will collaborate with its neighbors to determine whether there are enough events (within a predetermined time period) to warrant track processing and then to elect a node to aggregate detection events among neighbor nodes. The elected node will calculate track update information, determine whether current update is a continuation of a previously recognized track (data fusion), and send track update notices to nodes geographically located in the general direction of target travel. Track update notices are published and sent via diffusion to the collaborative signal processing applications running at neighbor nodes. Track update data are also moved into cache at a gateway node to be able to display it. In future implementations we would want the collaborative signal processing application to communicate directly with cache. However, for SenSoft v1, this integration could not be done and movement of track update data to a node gateway was accomplished by a special-purpose application at the gateway node that subscribed via diffusion to track updates and loaded all responses to gateway cache.

Target classification in SenSoft v1 is called as a function from the collaborative signal processing process. The PSU semantic information fusion (SIF)-based vehicle classification algorithm is called after data are received from a detection event. The routine requires (1) a pointer to a time series data block (in a BAE repository) and (2) start and stop times for the relevant data in the block. The routine then analyzes the data and returns, to the collaborative signal processing process, a success/failure indicator and a features structure: success/failure tells whether the routine was successful at classifying the target; features include vehicle locomotion (wheeled or tracked) and vehicle weight (light or heavy), along with confidence values for both. If classification is successful, then SIF also supplies a codebook value for the target. For accurate display, this codebook must match the one used by GUI processes.

The University of Maryland provides the task management software for interaction between a node gateway and a user system. The Maryland software consists of two basic process types: a ForkServer and

© 2005 by Chapman & Hall/CRC

a gateway. A ForkServer manages client (transmission control protocol/IP socket) connections with a gateway node, spawning a gateway process for each connection (i.e. linking it to a client socket). In this way, a gateway node can serve multiple simultaneous clients of varied types. Each gateway process maps tasking commands sent by a client into executions in the network to produce the information indicated by the commands. These executions would typically be queries and tasks for the on-node query/tasking component; however, in SenSoft v1 the tasks were executed simply as queries to the gateway node's cache to extract the desired data.

The SenSoft v1 GUI consists of a tactical display linked to a user-tasking interface through which users specify the kinds of data to be displayed. The tactical display, built by Maryland using BBN's OpenMap software [5], shows an overhead picture (zoomable) of the geographic area of interest enhanced by graphical depictions of many aspects of the underlying sensor network (e.g. sensor node position and payload, detection events, and fused target track plots). For example, a target track displays as an icon whose shape depends on the target classification (if available) and a "leader" vector (logarithmically proportionate to the target speed and pointing in the direction of travel). The interface currently paints sensor nodes as small green circles with pop-up information that includes the node's geographic coordinates, node network name, and payload. Detection events (when requested) are displayed as purple squares placed at the reported detection position (which is typically the reporting node's location). Figure 52.5 shows a GUI view of the sensor network testbed used for the SenSoft v1 demonstration. This view displays track plots indicating two targets (in this example, people) moving towards each other along the road. The testbed Web-cam view (ground truth) in the lower right corner of the display shows one of the targets. The detail pop-up boxes, highlighted along the left side, list the actual reported track values (classification, location, heading, velocity, and time) for each target.

The GUI is also used to specify queries and tasks to the network. Pop-up dialog boxes provided at the GUI allow the user to select the gateway conduit and specify a task (e.g. Display Events) and constraints (e.g. only at node 17) by choosing from task entry menus. The conduit then translates user (and system) requests into gateway commands, and moves (and deserializes) result data from the gateway into result objects that can be displayed by the GUI.

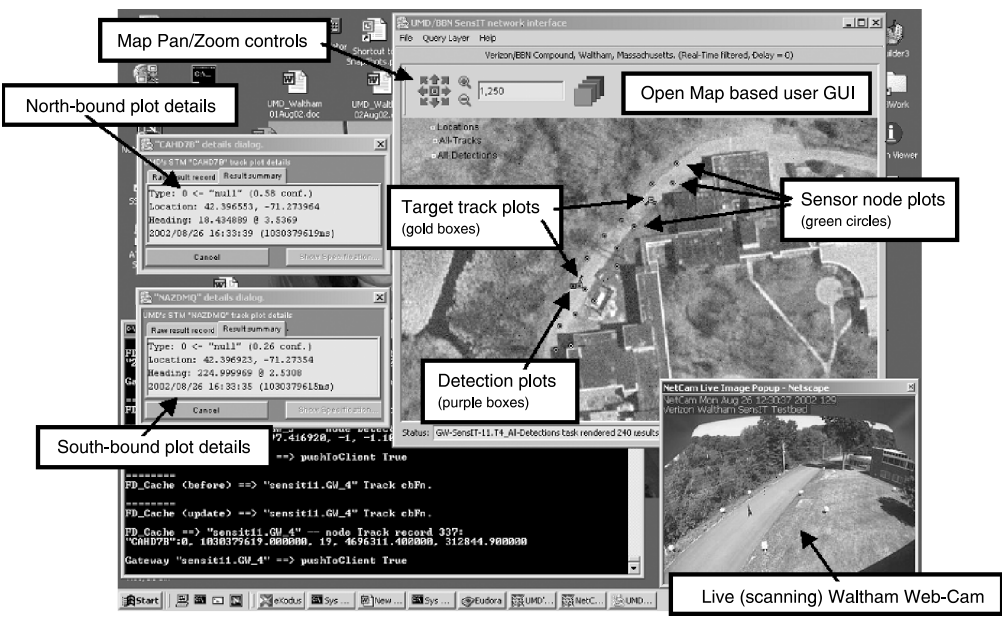


Figure 52.5. SenSoft GUI.

© 2005 by Chapman & Hall/CRC

52.9 Summary

The SensIT program encouraged a widely diverse research community to examine many issues pertaining to the information technology required to realize the vision of networked micro-sensors in tactical and surveillance applications. The program established the feasibility of creating networks of collaborating sensors, defined an initial architecture and developed an initial software prototype, and laid the groundwork for continued research in areas such as network management, collaborative detection, classification and tracking data routing and communications, and dynamic tasking and query.

The SenSoft architecture and its instantiation as SenSoft v1 provide a useful experimentation platform in support of research and development in distributed sensor networks. For example, this system could be used to support development and experimentation in collaborative signal processing for multiple and arbitrarily moving targets, comparisons and validations of costs for various communication and routing approaches, development of languages and other approaches to tasking and executing tasks within a network, or simulation and test of new tactical concepts for sensor systems, to name a few areas.

A next important step for SenSoft would be the (re-)design and implementation of well-defined interfaces (both syntactic and semantic definitions) for the infrastructure components. Such an experimentation platform would be an even stronger foundation for research and development, and a great step towards realization of a fieldable, operational SensIT system of fully distributed, autonomous, networked ground sensors.

References

[1] *WINS NG 2.0 User's Manual and API Specification*, Rev. A, May 30, 2002.

[2] Intanagonwiwat, C. *et al.*, Directed diffusion: a scalable and robust communication paradigm for sensor networks, in *Proceedings of the Sixth Annual International Conference on Mobile Computing and Networks (MobiCOM 2000)*, Boston, MA, August 2000.

[3] Silva, F. *et al.*, *Network Routing Application Programmer's Interface (API) and Walk Through 9.0.1*, December 9, 2002.

[4] Brooks, R.R. *et al.*, Self-organized distributed sensor network entity tracking, *International Journal of High Performance Computing Applications*, 16(3), 207–220, 2002.

[5] http://www.openmap.org (last accessed on July 2004).

© 2005 by Chapman & Hall/CRC

53

Statistical Approaches to Cleaning Sensor Data

Eiman Elnahrawy and Badri Nath

53.1 Introduction

Sensor networks have become an important source of information with numerous real-life applications. Existing networks are used for monitoring several physical phenomena, such as contamination level in soil and water, climate, building structure, habitat, and so on, potentially in remote harsh environments [1,2]. They have also found several interesting applications in industrial engineering and inventory management, such as monitoring the quality of perishable food items, as well as in transportation and traffic control [3,4].

Important actions are usually taken based upon the sensed information or sensor measurement; therefore, the quality, reliability and timeliness are extremely important issues in such applications. Data collected from wireless sensor networks, however, are subject to several problems and sources of errors. The imprecision, loss, and transience in wireless sensor networks, at least in their current form, as well as the current technology and the quality of the (usually very cheap) wireless sensors used, contribute to the existence of these problems. This implies that these networks must operate with imperfect or incomplete information [1,5]. These problems, however, may seriously impact the actual usage of such networks. They may yield imprecise or even incorrect and misleading answers to any query on sensor data. Therefore, online cleaning of sensor data before any decision making is crucial. In this chapter we will focus on probabilistic efficient and scalable approaches for this task. We shall discuss the following problems: reducing the effect of random errors, detection of outliers and malicious sensors, and recovery of missing values [6,7]. Before we proceed, let us discuss some examples to illustrate the significance of this problem.

Example 53.1. Becteria growth in perishable food items can be estimated by using either specialized sensors or by estimating it from temperature and humidity sensors attached to the items. Specialized sensors are quite expensive. On the other hand, temperature and humidity sensors are much cheaper and more cost effective. Therefore, the second alternative will usually be preferable. However, those cheap sensors are quite noisy, since they are liable to several sources of errors and environmental effects.

© 2005 by Chapman & Hall/CRC

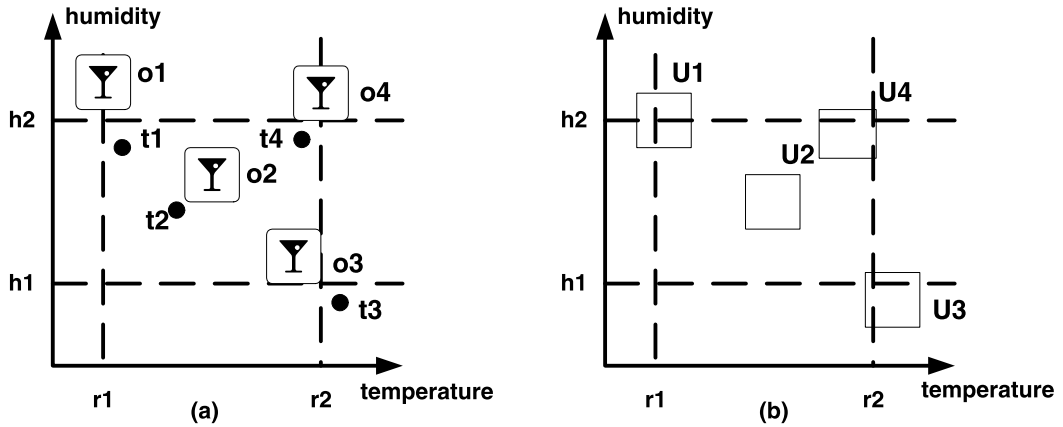


Figure 53.1. (a) Based on the observed readings items 1 and 4 will be thrown away. (b) Based on the uncertainty regions, only item 3 will be thrown away.

Consider the scenario of Figure 53.1(a), simplified for the sake of illustration. If the temperature and the humidity conditions of any item fall under or go over given thresholds, then the item should be thrown away. Assume that the range of acceptable humidity and temperature are $[h_1, h_2]$, and $[r_1, r_2]$ respectively. t_i refers to the true temperature and humidity readings at item i, and o_i refers to the reported (observed) readings at item i. As shown in the figure and based on the reported noisy data, items 1 and 4 should be thrown away, whereas items 2 and 3 should remain. However, based on the true readings, item 1 should remain and item 3 should be thrown away!

Example 53.2. Sensors become malicious and start reporting misleading unusual readings when their batteries are about to run out. Serious events in the areas monitored by the sensors could also happen. In this case, the sensors also start reporting unusual readings. Detecting and reasoning about such outliers in sensor data in both cases is, therefore, an important task.

Consider the scenario shown in Figure 53.2. Frame (a) shows random observations that are not expected in dense sensor networks. On the other hand, frames (b), (c) show two consecutive data samples obtained from a temperature-monitoring network. The reading of sensor i in frame (c) looks suspicious, given the readings of its neighbors and its own last reading. Intuitively, it is very unusual that the reading of i will "jump" from 58 to 40, from one sample to another. This suspicion is further strengthened with knowledge of the readings in the neighborhood. In order for sensor i to decide

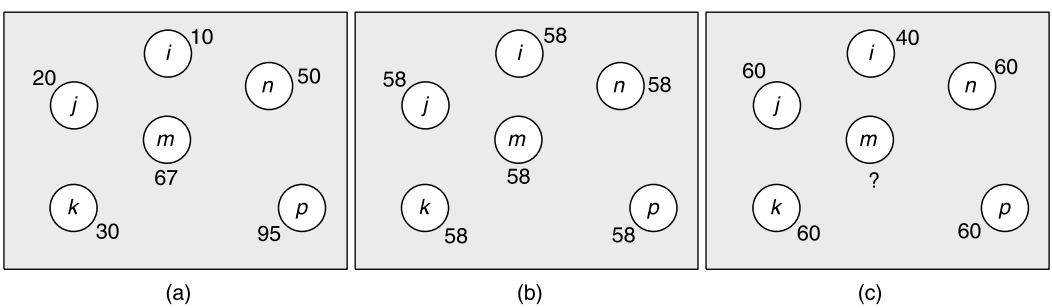


Figure 53.2. (a) Random observations that are not expected in dense sensor networks. (b), (c) Two consecutive data samples obtained from a dense network.

© 2005 by Chapman & Hall/CRC

whether this reading is an outlier, it has to know its most likely reading in this scenario. We shall show how we can achieve this later in this chapter.

53.2 Bayesian Estimation and Noisy Sensors

Random errors and noise that affect sensors result in uncertainty in determining the true reading or measurement of these sensors. Since these sensors are prone to errors they are uncertain about their true readings. Bayesian estimation can be utilized in order to reduce that effect, specifically by reducing the uncertainty associated with the noisy sensor data. Queries evaluated on the resultant clean and more accurate data are consequently far more accurate than those evaluated on the raw noisy data. It is important to notice that the reading of each individual sensor is usually important, i.e. fusion of readings from multiple sensors into one measurement to reduce the effect of noise is not usually an applicable or practical solution. Therefore, we apply Bayesian estimation to every sensor. Even if multiple sensor fusion is possible, we can apply the approach discussed below to enhance the accuracy of the result further.

The overall technique for cleaning and querying such noisy sensors is shown in Figure 53.3. It consists of two major modules: a cleaning module and a query-processing module. There are three inputs to the cleaning module: (1) the noisy observations reported from the sensors, (2) metadata about the noise characteristics of every sensor, which we call the error model, and (3) information about the distribution of the true reading at each sensor, which we call the prior knowledge. The output of the cleaning module is a probabilistic uncertainty model of the reading of each sensor which we call the posterior, i.e. a probability density function (pdf) of the true ''unknown'' sensor reading taking on different values.

The cleaning module is generally responsible for cleaning the noisy sensor data in an online fashion by computing accurate uncertainty models of the true ''unknown'' measurement. Specifically, it combines the prior knowledge of the true sensor reading, the error model of the sensor, and its observed noisy reading together, in one step and online using Bayes' theorem shown in Equation (53.1) (more information about Bayes' theorem can be found in [8–10]).

$$p(\theta|x) = \frac{\text{likelihood} \times \text{prior}}{\text{evidence}} = \frac{p(x|\theta)p(\theta)}{\int_{\psi} p(x|\Psi)p(\Psi)\mathrm{d}\Psi} \tag{53.1}$$

The likelihood is the probability that the data x would have arisen for a given value of the parameter θ and is denoted by $p(x|\theta)$. This leads to the posterior pdf of θ, $p(\theta|x)$.

The query-processing module is responsible for evaluating any posed query to the system using the uncertainty models of the current readings. Since the uncertainty models are probabilistic (i.e. describe random variables), traditional query-evaluation algorithms that assume a single value for each reading cannot be used. Hence, the query-processing step performs algorithms that are based on statistical

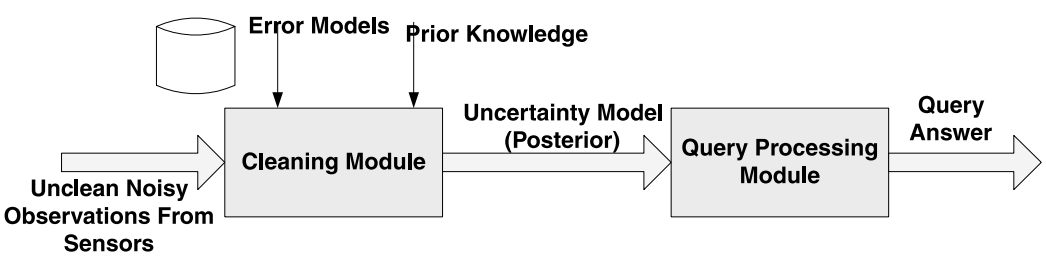


Figure 53.3. Overall framework.

© 2005 by Chapman & Hall/CRC

approaches for computing functions over random variables. A formal description of this overall technique is the topic of the next three sections.

There are two places where we can perform cleaning and query processing: at the sensor level or at the database level (or the base-station). Each option has its advantages and limitations in terms of its costs of communication, processing (which can be interpreted in terms of energy consumption), and storage. It is usually difficult to come up with explicit accurate cost models for each case, since there are many factors involved and some of them might be uncontrollable. In general, the overall system capabilities, sensors' characteristics, application, etc., will help us decide which option to choose. Some experimentation can also guide our final decision. A detailed discussion of these issues is beyond the scope of this chapter.

53.3 Error Models and Priors

There are numerous sources of random errors and noise in sensor data: (1) noise from external sources, (2) random hardware noise, (3) inaccuracies in the measurement technique (i.e. readings are not close enough to the actual value of the measured phenomenon), (4) various environmental effects and noise, and (5) imprecision in computing a derived value from the underlying measurements (i.e. sensors are not consistent in measuring the same phenomenon under the same conditions). The error model of each sensor is basically the distribution of the noise that affects it. We assume that it is Gaussian with zero mean. In order to fully define this Gaussian model we need to compute its variance. The variance is computed based on the specification of each sensor (i.e. accuracy, precision, etc.), and on testing calibrated sensors under normal deployment conditions. This testing can be performed either by the manufacturers or by the users after installation and before usage. Various environmental factors or characteristics of the field should also be taken into consideration. The error models may change over time, and new modified models may replace the old ones. Notice that non-Gaussian models can also be used, depending on the sensor's characteristics. The models, in general, are stored as metadata at the cleaning module. Sensors are not homogeneous with respect to their noise characteristics and, therefore, each sensor type, or even each individual sensor, may have its own error model.

Prior knowledge, on the other hand, represents a distribution of the true sensor reading taking on different values. There are several sources to obtain prior knowledge. It can be computed using facts about the sensed phenomenon, learning over time (i.e. history), using less noisy readings as priors for the noisier ones, or even by expert knowledge or subjective conjectures. They can also be computed dynamically at each time instance if the sensed phenomenon is known to follow a specific parametric model. For example, if the temperature of perishable items is known to drop by a factor of $x\%$ from time $t-1$ to time t then the cleaned reading of the sensor at time $t-1$ is used to obtain the prior distribution at time t. The resultant prior, the error model, and the observed noisy reading at time t are then input to the cleaning module in order to obtain the uncertainty model of the sensor at time t. Such a dynamic prior approach indeed resembles Kalman filters [14].

53.4 Reducing the Uncertainty

Let us assume that we have a set of n sensors in our network, $S = \{s_i\}$, $i = 1, \ldots, n$. These sensors are capable of providing their measurements at each time instance and reporting them to their base-station(s). Think of the reading of each sensor s at this instance as a tuple in the sensor database with attributes corresponding to its readings. Each sensor may have one or more attributes corresponding to each measurement. However, for simplicity of description, let us assume that each sensor measures a single phenomenon and that its measurement is real-valued. The following techniques can be fairly easily extended to accommodate multi-attribute and discrete-valued sensors.

Owing to occurrence of random errors the observed value of the sensor o will be noisy, i.e. it will be higher or lower than the true unknown value t. As we discussed in Section 53.3, the random error is Gaussian with zero mean and a known standard deviation $\sim N(0, \delta^2)$. Therefore, the true value t follows

© 2005 by Chapman & Hall/CRC

a Gaussian distribution centered around a mean $\mu = t$ and with variance δ^2, i.e. $p(o|t) \sim N(t, \delta^2)$. We apply Bayes' theorem to reduce the uncertainty and obtain a more accurate model which we call the posterior pdf for t, $p(t|o)$. We combine the observed value o, error model $\sim N(0, \delta^2)$, and the prior knowledge of the true reading distribution $p(t)$ as follows:

$$p(t|o) = \frac{p(o|t)p(t)}{p(o)} \tag{53.2}$$

This procedure is indeed generic. Although we explicitly assumed Gaussian errors, we do not have to restrict neither the error nor the prior distribution to a specific class of distributions (i.e. Gaussian). However, Gaussian distributions have certain attractive properties which makes them a good choice for modeling priors and errors. In particular, they yield another Gaussian posterior distribution with easily computed parameters as illustrated in the following example. This nice property enables performing the cleaning efficiently at the sensor level where we usually have restricted processing and storage. Moreover, Gaussian distributions are known to be analytically tractable, they are also useful for query processing and yield closed form solutions as we will show in the next Section. Nevertheless, they have the maximum entropy among all distributions [4]. Therefore, approximating the actual distribution for the error and the prior by suitable Gaussian distributions is usually advantageous.

Example 53.3. In order to understand how Bayesian estimation works, let us assume that the reading of a specific sensor s is known to follow a Gaussian distribution with mean μ_s and standard deviation σ_s, i.e. $t \sim N(\mu_s, \sigma_s^2)$, which is our prior. By applying Bayes' theorem and using some properties of the Gaussian distribution we can easily conclude that the posterior probability $p(t|o)$ also follows a Gaussian distribution $N(\mu_t, \sigma_t^2)$ [8,9]. Equations (53.3) and (53.4) show how the parameters of this posterior:

$$\mu_t = \frac{\delta^2}{\sigma_s^2 + \delta^2}\mu_s + \frac{\sigma_s^2}{\sigma_s^2 + \delta^2}o \tag{53.3}$$

$$\sigma_t^2 = \frac{\sigma_s^2 \delta^2}{(\sigma_s^2 + \delta^2)} \tag{53.4}$$

Why is this Bayesian approach superior? Suppose that we used a straightforward approach for modeling the uncertainty in sensor readings due to noise. That is, we will assume that the true unknown reading of each sensor follows a Gaussian pdf, centered around the observed noisy reading, with variance equal to the variance of the noise at this sensor, δ^2. Let us call this approach the no-prior approach. To prove the effectiveness of Bayesian estimation in reducing the uncertainty, let us consider the Bayesian mean-squared error, $E[(t - \hat{t})^2]$ for the resultant posterior with parameters μ_t and σ_t shown in Equations (53.3) and (53.4), where t and $\hat{t}$ are the true unknown reading and the posterior mean respectively.

Now let us compare it with the no-prior approach. The error or the uncertainty in the resultant posterior equals $\sigma_t^2 = \delta^2[\sigma_s^2/(\sigma_s^2 + \delta^2)]$ (refer to Kay [12] for the proof). This amount is less than δ^2, the error (or uncertainty) in the no-prior approach. Therefore, the Bayesian approach is always superior. Moreover, when the variance of the prior becomes very small compared with the variance of the noise (in other words, when the prior becomes very strong) the error of the posterior becomes smaller and the uncertainty is further reduced. Consequently, the Bayesian-based approach becomes far more accurate than the no-prior one. In general, if the prior knowledge is not strong enough, i.e. if it has a very wide distribution compared with the noise distribution, then the Bayesian-based approach will still be superior, though not "very" advantageous in terms of estimation error. Fortunately, in many situations this is not the case. For example, consider situations where we have cheap and very noisy sensors

© 2005 by Chapman & Hall/CRC

scattered everywhere to collect measurements of a well-modeled phenomenon such as temperature, etc. A strong prior can be easily computed while the noise is expected to have a very wide variance.

Equation (53.3) also illustrates an interesting fact. It shows that the Bayesian-based approach, in general, compromises between the prior knowledge and the observed noisy data. When the sensor becomes less noisy, its observed reading becomes more important and the model depends more on it. At very high noise levels the observed reading could be totally ignored.

53.5 Traditional Query Evaluation and Noisy Sensors

There are major differences between evaluation of queries over noisy sensor (uncertainty models) and exact data (single points). In uncertainty models, the reading of each noisy sensor at a specific time instance is considered a random variable (r.v.) described by the posterior pdf of that sensor and not necessarily by a single point with unit probability. Therefore, traditional query-evaluation algorithms that assume single points cannot be used for noisy sensors. Another significant difference is illustrated by the following example.

Example 53.4. Consider that we have noisy temperature sensors in our network. We would like to know the maximum reading of those sensors that record a temperature $\geq 50^{\circ}$F at a specific time instance. However, we do not have a single estimate of the true reading of each sensor, but rather we have a pdf that represents the "possible" values of that reading. In order to determine whether or not a specific sensor satisfies this predicate (i.e. a temperature $\geq 50^{\circ}$F), we have to compute the probability that each sensor satisfies the predicate using its posterior pdf. When the probability is less than 1, which is highly expected, we will be "uncertain" whether the sensor satisfies the predicate or not. Even though there is a high chance that a specific sensor satisfies the predicate as its probability approaches one, e.g. 0.8, neither the processing module nor any person can decide for sure. Therefore, there is no answer to this predicate and, consequently, we cannot decide which sensor reads the maximum temperature!

In order to overcome this difficulty without violating any statistical rules, we can modify our question by rephrasing it as "return the maximum value of those sensors that have at least a c% chance of recording at a temperature $\geq 50^{\circ}$F." We call c the "confidence level," and it is user defined as part of the queries. Following this reasoning, we can now filter out all those sensors that have a probability less than $c/100$ of satisfying our query and return the maximum of the remaining sensors. This leaves the problem of computing the maximum over a pdf, which we will discuss shortly.

Definition 53.1. The confidence level, or the acceptance threshold, c is a user-defined parameter that reflects the desired user's confidence. In particular, any sensor with probability $p < (c/100)$ of satisfying the given predicate should be excluded from the answer to the posed query.

53.6 Querying Noisy Sensors

Let us now discuss several algorithms for answering a wide range of traditional SQL-like database queries and aggregates over uncertain sensor readings. These queries do not form a complete set of all possible queries on sensors, but they do help illustrate the general approach to solving this problem. These algorithms are used in the processing module centrally at the database level, over the output of the cleaning module. They are generally based on statistical approaches for computing functions over one or more random variables. For simplicity of notation, we will use the term $p_{s_i}(t)$ to describe the uncertainty model $p(t|o)$ of sensor s_i.

53.6.1 Class I

The first class of queries returns the value of the attribute of the queried sensor (i.e. its reading). A typical query of this class is "What is the reading of sensor x?" There are two approaches for evaluating

© 2005 by Chapman & Hall/CRC

this class of queries. The first one is based on computing the expected value of the probability distribution it as follows:

$$E_{s_i}(t) = \int_{-\infty}^{\infty} t p_{s_i}(t)\,\mathrm{d}t \tag{53.5}$$

where s_i is the queried sensor.

The second approach is based on computing the p% confidence interval of $p_{s_i}(t)$. The confidence factor p, ($p = c/100$) is user defined with a default value equal to 95. The confidence interval is computed using Chebychev's inequality [13] as follows:

$$P(|t - \mu_{s_i}| < \epsilon) \geq 1 - \frac{\sigma_{s_i}^2}{\epsilon^2} \tag{53.6}$$

where μ_{s_i} and σ_{s_i} are the mean and the standard deviation of $p_{s_i}(t)$, $\epsilon > 0$. In order to compute ϵ we set $[1 - (\sigma_{s_i}^2/\epsilon^2)]$ to p and solve. The resultant p% confidence interval on the attribute will be $[\mu_{s_i} - \epsilon, \mu_{s_i} + \epsilon]$.

53.6.2 Class II

This class of queries returns the set of sensors that satisfy a predicate. A typical query of this class is "Which sensors have at least c% chance of satisfying a given range?" The range $R = [l, u]$ is specified by lower and upper bounds on the attribute values l and u respectively. The answer to this class is the set $S_R = \{s_i\}$ of those sensors with probability ($p_i \geq (c/100)$) of being inside the specified range R, where $p_i = \int_l^u p_{s_i}(t)\,\mathrm{d}t$ along with their "confidence" p_i. Although this is a simple range query, the algorithm extends naturally to more complex conditions with mixes of AND and OR as well as to the multi-attribute case.

Example 53.5. Consider the scenario of Figure 53.1(b) where we have sensors of two attributes. Assume that the output of the cleaning module is that the reading of each sensor is uniformly distributed over the depicted squared uncertainty regions. The probabilities of the items being inside the given range are (item1, 0.6), (item2, 1), (item3, 0.05), (item4, 0.85). If the user-defined confidence level is $c = 50\%$, which is a reasonable confidence level, then only item 3 will be thrown away. This coincides with the correct answer over the true unknown readings, and is also more accurate than the answer on the noisy (uncleaned) readings.

53.6.3 Class III

The last class of queries that we consider is aggregate queries of the form "On those sensors which have at least c% chance of satisfying a given predicate, what is the value of a given aggregate?" Before evaluating the aggregate, we obtain the set S_R of those sensors that satisfy the given predicate using the algorithm of Class II. If the predicate is empty then all sensors in the network are considered in the aggregation, i.e. $S_R = S$. In general, the aggregate can be a summary aggregate such as SUM, AVG, and COUNT aggregates or an exemplary aggregate such as MIN, MAX aggregates (this classification of aggregate queries into summary and exemplary has been used extensively among the database community) [2].

To compute the SUM aggregate we utilize a statistical approach for computing the sum of independent continuous random variables, also called convolution. To sum $|S_R|$ sensors each represented by a r.v., we perform the convolution on two sensors and then add one sensor to the resultant sum, which is also a r.v., repeatedly till the overall sum is obtained. Sepecifically assume that the sum $Z = s_i + s_j$ of two uncertainty models of sensors s_i and s_j is required. If the pdfs of these two

© 2005 by Chapman & Hall/CRC

sensors are $p_{s_i}(t)$ and $p_{s_j}(t)$ respectively, then the pdf of Z is computed using Equation (53.7) [13]. The expected value of the overall sum or a 95% confidence interval can then be computed and output as the answer similar to Class I:

$$p_Z(z) = \int_{-\infty}^{\infty} p_{s_i}(x) p_{s_j}(z - x)\, dx \tag{53.7}$$

Computing the COUNT query reduces to output $|S_R|$ over the given predicate. The answer of the AVG query however equals the answer of the SUM query divided by the answer of the COUNT query, over the given predicate. On the other hand, the MIN of m sensors in S_R is computed as follows. Notice that the MAX query is analogous. Nevertheless, other order statistics such as Top-K, Min-K, and median can be computed in a similar manner.

Let the sensors $s_1, s_2, \ldots, s_m$ be described by their pdfs $p_{s_1}(t), \ldots, p_{s_m}(t)$, respectively, and their cumulative distribution functions (cdfs) $P_{s_1}(t), \ldots, P_{s_m}(t)$, respectively. Let the random variable $Z = \min(s_1, s_2, \ldots, s_m)$ be the required minimum of these independent continuous r.vs. The cdf, pdf of Z, $P_Z(z)$, $p_Z(z)$ are computed using Equations (53.8), (53.9), respectively [3].

$$\begin{aligned} P_Z(z) =& \text{prob}(Z \leq z) = 1 - \text{prob}(Z > z) \\ =& 1 - \text{prob}(s_1 > z_1, s_2 > z, \ldots, s_m > z) \\ =& 1 - (1 - P_{s_1}(z)) \cdots (1 - P_{s_m}(z)) \end{aligned} \tag{53.8}$$

$$p_Z(z) = -\frac{d}{dx}(1 - P_{s_1}(z))(1 - P_{s_2}(z)) \cdots (1 - P_{s_m}(z)) \tag{53.9}$$

53.6.4 Approximating the Integrals

The above algorithms involve several integrals that are not usually guaranteed to yield a closed-form solution for all families of distributions. We recommended Gaussian priors and error models in Section 53.4. Here is another motivation for this recommendation. There are specific formulas for computing these integrals easily in the case of Gaussian distributions. For example, the marginal pdf of a Gaussian is also a Gaussian, so is the sum of Gaussians (and consequently the AVG) [13]. Evaluation of Class I queries simply reduces to the mean parameter μ of the Gaussian uncertainty model in the single attribute case, and to the m-component mean vector in the multi-attributes case.

For other families of distributions where no known closed-form solution exists, we can approximate the integrals by another suitable distribution. We then store these approximations in a repository at the query-processing module. Therefore, a large part of the computation is performed offline and reused when needed, e.g. by changing the parameters in precomputed parametric formulas.

53.7 Spatio-Temporal Dependencies and Wireless Sensors

In the previous sections we showed how Bayesian estimation and statistics are used for reducing the effect of noise in noisy sensors and for querying them. In the rest of this chapter we discuss a statistical approach for detecting malicious sensors or serious anomalies in sensor data efficiently and online, as well as recovering missing readings. This approach is based on exploiting the spatio-temporal dependencies in sensor networks.

Sensor networks are usually dense for coverage and connectivity purposes, robustness against occlusion, and for tolerating network failures [14–17]. These dense networks are usually used for monitoring several well-defined real-life phenomena where redundant and correlated readings, specifically spatio-temporally, exist. In particular, there are spatial dependencies between spatially

© 2005 by Chapman & Hall/CRC

adjacent sensor nodes, as well as temporal dependencies between history readings of the same node. Such dependencies, if defined appropriately, can enable the sensors to "predict" their current readings locally knowing both their own past readings and the current readings of their neighbors. This ability, therefore, provides a tool for detecting outliers and recovering missing readings.

Spatio-temporal dependencies can indeed be modeled and learned statistically using a suitable classifier, specifically by the use of a Bayesian classifier. The learning process reduces to learning the parameters of the classifier while the prediction process reduces to making inferences. The learning is performed online in a scalable and energy-efficient manner using in-network approaches that have been shown to be energy efficient theoretically and experimentally [2,18,19]. The inference in Bayesian classifiers is also straightforward and is afforded by the current generation of wireless sensors since it requires simple calculations. It is important, however, to notice that this solution is, in general, suitable to some classes of networks and applications where high spatio-temporal characteristics exist and can be learned, e.g. networks used for monitoring temperature, humidity, etc., and for tracking.

53.8 Modeling and Dependencies

Spatio-temporal data is a category of structured data. Precise statistical models of structured data that assume correlations between all observations are, in general, complicated and difficult to work with and, therefore, not used in practice. This is specifically due to the fact that they are defined using too many parameters that cannot easily be learned or even explicitly quantified. For example, to predict the reading of a sensor at a specific time we need to know the readings of all its neighbors, the entire history of its reading, and the parameters or the probabilistic influence of all these readings on its current reading. Alternatively, Markov-based models that assume "short-range dependencies" have been used in the literature to solve this difficulty [9,20]. Spatio-temporal dependencies in wireless sensor networks are modeled using the Markov assumption as follows: the influence of all neighboring sensors and the entire history of a specific sensor on its current reading is completely summarized by the readings of its immediate neighbors and its last reading. In other words, the features of classification (prediction) are (1) the current readings of the immediate neighbors (spatial), and (2) the last reading of the sensor (temporal) only.

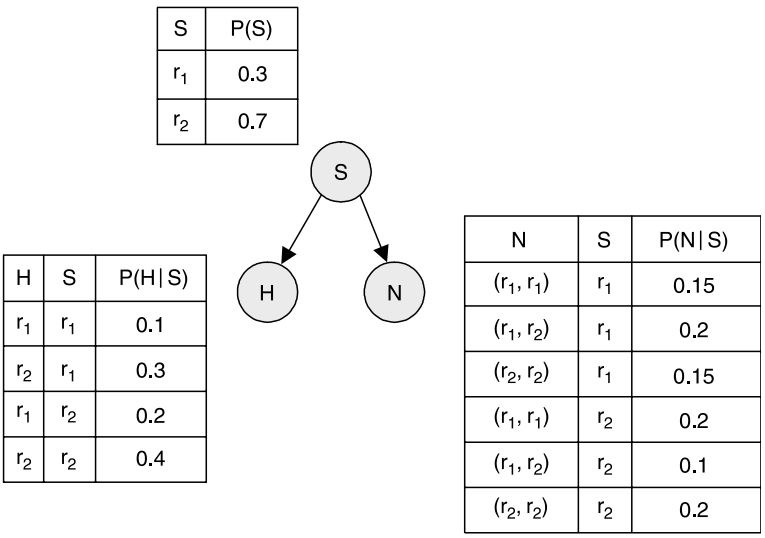
The Markov assumption is very practical. It drastically simplifies the modeling and significantly reduces the number of parameters needed to define Markov-based models.

Without loss of generality, assume that sensor readings represent a continuous variable that takes on values from the interval $[l, u]$. We divide this range $(u–l)$ into a finite set of m nonoverlapping subintervals, not necessarily of equal length, $R = \{r_1, r_2, \ldots, r_m\}$. Each subinterval is considered a class. These classes are *mutually exclusive* (i.e. nonoverlapping) and *exhaustive* (i.e. cover the range of all possible readings). R can be quantized in different ways to achieve the best accuracy and for optimization purposes, e.g. to make frequent or important intervals shorter that infrequent ones. The classifier involves two features: the history H and the neighborhood N. The history represents the last reading of the sensor, while the neighborhood represents the reading of any two nearby sensors. H takes on values from the set R, while N takes on values from $\{(r_i, r_j) \in R \times R, i \leq j\}$, and the target function (class value) takes on values from R.

Figure 53.4 shows the structure of the Bayes-based model used for modeling the spatio-temporal dependencies. N is a feature that represents readings of neighboring nodes, H is a feature that represents the last reading of the sensor, and S is the current reading of the sensor. The different values of sensor readings represent the different classes. The parameters of the classifier are the dependencies, while the inference problem is to compute the most likely reading (class) of the sensor given the parameters and the observed correlated readings.

We model the spatial information using readings from "some" of the neighboring nodes, the exact number varies with the characteristics of the network and the application. Notice that the continuously changing topology and common node failures in wireless networks prohibit any assumption about a specific spatial neighborhood, i.e. the immediate neighbors may change over the time. The model below

© 2005 by Chapman & Hall/CRC



S	P(S)
r_1	0.3
r_2	0.7

H	S	P(H|S)
r_1	r_1	0.1
r_2	r_1	0.3
r_1	r_2	0.2
r_2	r_2	0.4

N	S	P(N|S)
(r_1, r_1)	r_1	0.15
(r_1, r_2)	r_1	0.2
(r_2, r_2)	r_1	0.15
(r_1, r_1)	r_2	0.2
(r_1, r_2)	r_2	0.1
(r_2, r_2)	r_2	0.2

Figure 53.4. The Bayes-based model for spatio-temporal dependencies.

is fairly generalizable to any number of neighbors as desired. However, a criterion for choosing the neighbors that yield the best prediction, if information about all of them are available at the sensor, is an interesting research problem. In our discussion, let us assume a neighborhood that consists of two randomly chosen indistinguishable neighbors.

To define the parameters of this model, let us first show how the inference is performed. The Bayes classifier is generally a model for probabilistic inference; the target class, r_{NB}, output by the classifier is inferred probabilistically using maximum *a posteriori* (MAP) [9,21,22].

$$r_{\mathrm{MAP}} = \mathrm{argmax}_{r_i \in R} P(r_i|h, n) \tag{53.10}$$

where h and n are the values of H and N respectively. This can be rewritten using Bayes rule as follows.

$$r_{\mathrm{MAP}} = \mathrm{argmax}_{r_i \in R} \frac{P(h, n|r_i)P(r_i)}{P(h, n)} = \mathrm{argmax}_{r_i \in R} P(h, n|r_i)P(r_i) \tag{53.11}$$

Since the denominator is constant for all the classes, it does not affect the maximization and can be omitted. From this formula we see that the terms $P(h, n|r_i)$ and $P(r_i)$ should be computed for each h, n, r_i, i.e. they constitute the parameters of the model. To cut down the number of training data needed for learning these parameters and, consequently, to optimize and resources of the network, we utilize the ''naive Bayes'' assumption: the feature values are conditionally independent given the target class. That is, we assume that the spatial and the temporal information are conditionally independent given the reading of the sensor. This assumption does not sacrifice the accuracy of the model. Although it is not true in general, ''naive'' Bayes classifiers have been shown to be efficient in several domains where this assumption does not hold, even competing with other, more sophisticated classifiers [9,21,22]. Based on this conditional-independence assumption, we obtain the following

$$r_{\mathrm{NB}} = \mathrm{argmax}_{r_i \in R} P(r_i)P(h|r_i)P(n|r_i) \tag{53.12}$$

© 2005 by Chapman & Hall/CRC

The parameters now become (a) the two Conditional Probabilities Tables (CPTs) for $P(h|r_i)$ and $P(n|r_i)$, and (b) the prior probability of each class $P(r_i)$. These parameters model the spatio-temporal dependencies at each sensor in the network and enable it to predict its reading.

53.9 Online Distributed Learning

The spatio-temporal dependencies or the parameters of the Bayesian classifier are learned from training sensor data in a distributed in-network fashion. The training data are the triples (h, n, r_t), where h represents the last reading of the sensor, n represents the current reading of two neighbors, and r_t represents the current reading of that sensor. This training information is available at each node when sampling the network, since the shared channel enables ''snooping'' on neighbors broadcasting their readings. The snooping should be performed correctly in order to incorporate a very little cost and no communication complications. The neighbors can for example, be the parent of the sensor and one of its children in the case of a routing tree. To account for the lack of synchronization, the node quantizes the time, caches the readings of the neighbors over each time slot, caches its own last reading, and uses them for learning at the end of the slot. If at any slot the training instance is not complete, i.e. some information is missing, this training instance is discarded and not used for the learning phase.

If the sensed phenomenon is completely nonstationary and coupled to a specific location, i.e. if the dependencies are coupled with a specific location, then the parameters are learned as follows. Each node estimates $P(r_i)$, $i = 1, \ldots, m$, simply by counting the frequency with which each class r_i appears in its training data, i.e. its sensed value belongs to r_i. The node does not need to store any training data to perform this estimation; it just keeps a counter for each r_i and an overall counter of the number of instances observed so far, all initialized to zero. It increments the appropriate counter whenever it observes a new instance. The CPTs of H and N are estimated similarly. Notice that $P(H = h|r_i)$ is the number of times that ($H = h$ AND the sensor reading belongs to r_i) divided by the number of times the class is r_i. Since the node already keeps a counter for the latter, all it needs is a counter for each ($H = h$ AND the reading belongs to r_i), a total of m^2 counters. In order to obtain the CPT for $P(n|r_i)$, in the case of two indistinguishable neighbors, the node keeps $m^2(m+1)/2$ counters for each $(n = (r_i, r_j), i = 1, \ldots, m, j = 1, \ldots, m, i \leq j$ AND the sensor reading belongs to $r_i)$ since (r_i, r_j) is indistinguishable from (r_j, r_i). That is a total of $1 + m + \frac{3}{2}m^2 + \frac{m^3}{2}$ counters are needed. After a predefined time interval, a testing phase begins where the sensor starts testing the accuracy of the classifier. It computes its predicted reading using the learned parameters at each time slot and compares it with its sensed reading. It keeps two counters: the number of true predictions and the number of false predictions. At the end of the testing phase, the sensor judges the accuracy by computing the percentage of the correctly classified test data. If the accuracy is not acceptable according to a user-defined threshold, then the learning resumes. The sensor repeats until the accuracy is reached or the procedure is terminated by the base-station.

On the other hand, if the phenomenon being sensed is not stationary over the time, then the sensors relearn the parameters dynamically at each change. They can be preprogrammed so that they relearn at specific predefined time instances or they can detect the changes dynamically, e.g. when the error rate of the previously learned parameters increases significantly. In both cases, the old learned correlations can be stored at the base-station and reused if the changes are periodic. The sensors periodically send their parameters to the base-station to be recovered if they fail.

If the sensed phenomenon is stationary over the space, then the above learning procedure is modified to scale with the size of the network in an energy-efficient fashion using in-network aggregation. Specifically, the locally learned parameters at each node are combined together by ''summing'' the individually learned counters over all the nodes via in-network aggregation (SUM aggregate). So are the testing parameters (the two counters). The final overall parameters are then used by each sensor in the network. Notice that stationarity in space does not imply a static or a fixed topology. It rather implies that we can use training data from all the nodes to learn the parameters. Therefore, it enables the collecting of a large number of training instances in a relatively short time. The approach also adapts

© 2005 by Chapman & Hall/CRC

to less perfect situations where the "stationarity in space" holds inside clusters of sensors over the geographic space. A separate model is learned and then used by the sensors in each region. Finally, a hybrid approach for nonstationary phenomena in time and space is formed in a fairly similar way. In all cases, Bayes-based models converge using a small training set [21]. The convergence also makes it insensitive to common problems such as outliers and noise, given that they are usually random and infrequent, and duplicates, since the final probabilities are the ratio of two counters.

A centralized approach for learning, where the parameters are learned centrally at the base-station, can be used in some situations. However, it is sometimes inferior to in-network learning with respect to its communication cost, which is the major source of power consumption in sensor networks [2,15,16]. Moreover, in-network learning effectively reduces the number of forwarded packets, which is a serious disadvantage of the centralized learning. In general, this decision is application-dependent and is driven by various factors, such as the size of the training data.

To understand the trade-offs between the distributed and the centralized approach, consider a completely nonstationary (in space) network, where learning is performed at each node, a centralized approach is inferior due to obvious communication cost. For stationary or imperfectly stationary networks the trade-off is not that clear. We notice that in-network learning involves computing of a distributive summary aggregate, while centralized learning can be viewed as computing of a centralized aggregate or as collecting of individual readings from each node [2]. Therefore, assuming a fairly regular routing tree, the communication cost of in-network learning is roughly $k \times O(m^3) \times O(n)$, where k is the number of epochs, m is the number of classes, and n is the number of nodes used for learning the parameters, which can be as large as the size of the network. This is equivalent to the cost of computing $O(m^3)$ summary aggregates k times. The cost of a centralized learning is roughly $p \times O(n^2)$, where p is the size of training data at each sensor which is "application dependent." This is equivalent to the cost of computing p centralized aggregates (a detailed analysis of the cost of computing aggregates using in-network aggregation can be found in [2]). It has been shown that is yields an order of magnitude reduction in communication over centralized approaches. For a realistic situation, where $p = 1000$, $k = 2$, $m = 5$, $n = 10$, the cost of centralized learning is an order of magnitude higher. This difference further increases for perfectly stationary situations, since n becomes very large. Even when m increases the difference remains significant. The above analysis fairly easily extends to the case of nonstationarity in time.

53.10 Detecting Outliers and Recovery of Missing Values

The Bayesian classifier can be used for inference once its parameters are learned. In particular, the probability of a sensor reading taking on different values, i.e. being in different classes, r_i, $i = 1, \ldots, m$, is computed for every r_i from Equation (53.13) using the learned parameters, current readings of neighbors n, and the last reading of this sensor h. The class with highest probability is then output as the prediction:

$$P(r_i|h, n)\, P(r_i)P(h|r_i)P(n|r_i) \tag{53.13}$$

Example 53.6. Consider the scenario shown in Figure 53.2. Assume that sensor readings can take values in the range [30, 60]. Assume that we divided this range into two classes, $r_1 = [30, 45]$ and $r_2 = [45, 60]$. Further assume that we have already learned the parameters, i.e. the CPTs, shown in Figure 53.4. To infer the missing reading of sensor m in frame (c), we use the readings of sensors j and k in this frame, the history of m, $H = r_2$. We compute $P(r_1|h = r_2, n = (r_2, r_2)) \sim 0.3 \times 0.3 \times 0.15 = 0.0135$, while $P(r_2|h = r_2,\ n = (r_2,\ r_2)) \sim 0.7 \times 0.4 \times 0.2 = 0.056$. According to this, the second class is more likely. This indicates that the reading of sensor m is expected to be somewhere in the range [45, 60].

The ability of the sensor to predict its reading is a very powerful data-cleaning tool; it is used for detecting false outliers/serious anomalies, and approximating its reading when missing. To utilize the Bayesian model in outlier detection, the sensor "locally" computes the probability of its reading being in different classes using Equation (53.13). It then compares the probability of its most likely reading,

© 2005 by Chapman & Hall/CRC

i.e. highest probability class, and the probability of its actual sensed reading. If the two differ significantly then the sensor may decide that its reading is indeed an outliner. For example, we follow the steps of Example 53.6 to compute the probability of sensor i taking on values in the ranges [30, 45] and [45, 60]. We find that its reported reading, i.e. 40, in Figure 53.2, is indeed an outlier, since the probability of its reading being in [30, 45] $\sim$ 0.0135 is small compared with [45, 60] $\sim$ 0.056. Distinguishing anomalies from malicious sensors is somewhat tricky. One approach is to examine the neighborhood of the sensor at the base-station. In particular, if many correlated sensors in the same neighborhood reported alert messages then this is most likely a true serious anomaly.

The classifier is also used to recover missing values. The objective is to predict the missing reading of a specific sensor, which is performed by inferring its class using Equation (53.12). The predicted class represents a set of readings (a subinterval) and not a single specific value. We can, for example, choose the median of this subinterval as the predicted single reading; therefore, the error margin in prediction becomes less than half the class width. Think of this approach as significantly reducing the uncertainty associated with the missing reading from $[l, u]$ to r_i, where $[l, u]$ is the interval of all possible readings, while r_i is a small subinterval of $[l, u]$. As the width of each class becomes smaller, so the uncertainty decreases further. In general, there is a trade-off between the complexity of the classifier and the uncertainty. Smaller subintervals translate to more classes and, consequently, to bigger CPTs, which are hard to work with and to store locally at the sensor. Therefore, the width of "each" of the classes is chosen wisely; we assign small classes to important readings that require tight error margins.

Recovery of missing values can be generalized to in-network sampling where significant energy is saved. We "strategically" select a subset of the sensors to sense the environment at a specific time while predicting the missing readings within acceptable error margins, i.e. we perform a sampling. The selection criteria are based on the geographical locations, remaining energy, etc. A complete re-tasking of the entire network can be performed, e.g. when some sensors are about to run out of battery then their sampling rate is reduced, and so on. A basic algorithm is to control the nodes in a way such that they alternate sensing the environment by adjusting their sampling rate appropriately.

53.11 Future Research Directions

We discussed probabilistic approaches for solving some cleaning problems in sensor networks online. There are several challenges and open problems in this area that need further investigation. Wireless sensors are becoming very pervasive. New applications are emerging every day that rely on these sensors for decision making. Therefore, quality and integrity of sensor data are very important problems. The future of wireless sensors lies in reasoning about and solving these data-cleaning problems "efficiently," in terms of the available resources, and "online." Existing research has always focused on providing a low-level networking solution or customized solutions that work for specific applications [1,2]. In both cases, these problems persist, though less severely. Hence, general-purpose solutions are needed.

We discussed only simple, traditional database queries in this chapter. In most of the algorithms the query evaluation was centralized. A distributed version of these algorithms, as well as addressing more complicated queries and optimization issues, is an interesting research direction. It is important that the accuracy of the devised algorithms be suitable to the application at hand.

Generalization to sampling and to heterogeneous sensors are challenging problems. Readings obtained from a dense sensor network are sometimes highly redundant. In some cases they may be complementary to each other. Therefore, queries can be evaluated on a sample of the sensors only. A large part of existing work on query processing in sensor networks has only focused on homogeneous clean data from all sensors [2,23,24]. However, sensors may not be homogeneous. They usually differ in their remaining energy, storage, processing, and noise effect. A repository is therefore needed at the database system to store metadata about the capabilities and the limitations of each sensor. The database system should be able to turn the sensors on/off or control their rate using proxies [23]. The underlying networking functionality should allow for such a scenario. Users may also define specific quality requirements on the answer to their queries as part of the query, e.g. a confidence level,

© 2005 by Chapman & Hall/CRC

the number of false positives/negatives, etc. The challenge is how to minimize the number of redundant sensors used unnecessarily to answer a specific query while (1) meeting the given quality level (e.g. confidence) and (2) ''best'' utilizing the resources of the sensors. The sample size may need to be increased or specific, more accurate sensors may have to be turned on in order to meet the given user's expectations. The sampling methods may have to be changed over the time (random, systematic, stratified, etc.). In general, this introduces another cost factor in decision making and actuation, query optimization and evaluation, and resource consumption. This problem is also related to the Bayesian classifiers. It is important to investigate the optimal number of neighbors needed and the effect of selecting the neighbors intelligently versus randomly on the accuracy of prediction. Several real deployment decisions in the approaches discussed in this chapter are application dependent. Experimentation and characterization are needed for guiding such decisions.

Handling noise in one-dimensional sensors (i.e. sensors with single attirubtes) can be easily extended to the multi-dimensional case. However, handling multi-dimensional outliers and missing values is far more complicated and is still an open problem. So is extending the quantized Bayesian classifier to the case of continuous classes. Finally, it is interesting to investigate non-Bayesian solutions to the problems discussed, as well as cross-evaluation of these solutions with Bayesian-based ones.

References

[1] Zhao, J. *et al.*, Computing aggregates for monitoring wireless sensor networks, in *Proceedings of IEEE SNPA'03*, 2003.

[2] Madden, S. *et al.*, TAG: a tiny aggregation service for *ad-hoc* sensor networks, in *Proceedings of 5th Annual Symposium on Operating Systems Design and Implementation (OSDI)*, December 2002.

[3] Bonnet, P. *et al.*, Towards sensor database systems, in *Proceedings of the Second International Conference on Mobile Data Management*, January 2001.

[4] Wolfson, O. *et al.*, The geometry of uncertainty in moving objects databases, in *Proceedings of International Conference on EDBT*, 2002.

[5] Ganesan, D. *et al.*, Highly-resilient, energy-efficient multipath routing in wireless sensor networks, *Mobile Computing and Communications Review (MC2R)*, 5(4), 2002.

[6] Elnahrawy, E. and Nath, B., Cleaning and querying noisy sensors, in *Proceedings of the Second ACM International Workshop on Wireless Sensor Networks (WSNA'03)*, 2003.

[7] Elnahrawy, E. and Nath, B., Context-aware sensors, in *Proceedings of first IEEE European Worshop on Wireless Sensor Networks* (*EWSN*), volume 2920, *Lecture Notes in Computer Sciences* (*LNCS*), Springer-verlag, Berlin, Heidelberg, 2004, 77.

[8] Box, G.E.P. and Tiao, G.C., *Bayesian Inference In Statistical Analysis*, Addison-Wesley, 1973.

[9] Hand, D. *et al.*, *Principles of Data Mining*, MIT Press, 2001.

[10] Duda, R.O. *et al.*, *Pattern Classification*, 2nd ed., John Wiley, 2001.

[11] Lewis, F.L., *Optimal Estimation: With an Introduction to Stochastic Control Theory*, John Wiley, 1986.

[12] Kay, S., *Fundamentals of Statistical Signal Processing, Volume I: Estimation Theory*, Prentice Hall, 1993.

[13] Casella, G. and Berger, R.L., *Statistical Inference*, Duxbury Press, Belmont, CA, 1990.

[14] Mainwaring, A. *et al.*, Wireless sensor networks for habitat monitoring, in *ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02)* 2002.

[15] Ganesan, D. and Estrin, D., DIMENSIONS: why do we need a new data handling architecture for sensor networks? in *Proceedings of Ist Workshop on Hot Topics in Networks (Hotnets-1)*, October 2002.

[16] Pottie, G. and Kaiser, W., Embedding the internet: wireless sensor networks, *Communications of the ACM 43* (5), 51, 2000.

[17] Liu, J. *et al.*, A dual-space approach to trackling and sensor manageent in wireless sensor networks, in *Proceedings of WSNA'02*, 2002.

© 2005 by Chapman & Hall/CRC

[18] Krishanamachari, B. *et al.*, The impact of data aggregation in wireless sensor networks, in *International Workshop of Distributed Event Based Systems (DEBS)*, July 2002.
[19] Heidemann, J. *et al.*, Building efficient wireless sensor networks with low-level naming, in *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, October 2001.
[20] Shekhar, S. and Vatsavai, R.R., Spatial data mining research by the Spatial Database Research Group, University of Minnesota, The Specialist Meeting on Spatial Data Analysis Software Tools, CSISS, and NSF Workshop on Spatio-Temporal Data Models for Biogeophysical Fields, 2002.
[21] Mitchell, T., *Machine Learning*, McGraw Hill, 1997.
[22] Witten, I.H and Frank, E., *Data Mining: Practical Machine Learning Tools and Techniques with JAVA Implementations*, Morgan Kaufmann, 2000.
[23] Madden, S. and Franklin, M.J., Fjording the stream: an architecture for queries over streaming sensor data, in *Proceedings of ICDE*, 2002.
[24] Hellerstein, J.M. *et al.*, Beyond average: towards sophisticated sensing with queries, in *Proceedings of IPSN'03*, 2003.

© 2005 by Chapman & Hall/CRC

54

Plant Monitoring with Special Reference to Endangered Species

K.W. Bridges and Edo Biagioni

54.1 Introduction

The monitoring of populations of endangered plants is a model system that provides a focused challenge to our development of integrated sensor and remote network technologies, operations, and interpretation. The concrete problems faced in the design, construction, and maintenance of such a system not only helps solve an urgent problem, but also provides a general test bed that applies to many situations based on distributed sensor networks.

Many plant species are at risk of becoming extinct. These endangered populations are found through out the world and occur in a wide range of habitats. While some of these rare species are being monitored, most receive only cursory attention. Put simply, we know little about the biology of many of these species, particularly how they respond to environmental conditions.

The general objective of plant monitoring is to acquire a significant time series of data about individual plants, populations of the species, or plant communities comprised of many species. In addition, a similar time sequence of environmental information is almost always gathered. Together, these data allow correlations between the plant life history events and the weather.

The plant life history events are called the ''phenology'' of the plant [1]. There is generally a set of phenological stages through which a plant grows, including the seed, seedling, juvenile, subadult, and adult stages. Within these stages, other phenological events are recognized, such as periods of growth, flowering, leaf flushing, and leaf fall. Different species, different habitats, and different environmental conditions sometimes require adjustments to these general phenological stages and events. It is not just scientists who monitor plant phenology. The advance of the fall colors as deciduous trees prepare to drop their leaves is a widely anticipated and closely monitored annual event by the entire population living in areas where this occurs. One of the remarkable properties of plant phenology, in general, is the close correlation to the local weather.

The weather information that is usually collected in phenological studies includes the air temperature, rainfall amount, solar radiation intensity, and relative humidity. Wind speed and direction are sometimes included in the set of measurements.

The emphasis in this chapter is on making observations of rare and endangered plant species and their surrounding environment. While this represents the general requirements of plant monitoring,

© 2005 by Chapman & Hall/CRC

it adds some additional constraints that will be discussed later. The value of choosing this special group of plants is that such monitoring may be essential to saving and recovering these species. This is an urgent need and one that is, unfortunately, very poorly served with our current technology. This emphasizes that this problem is interesting not only from an engineering perspective, but also has great social value. Any progress in solving the monitoring problems will help in a large number of general situations and may also be critical to our properly maintaining part of our biological heritage.

In the U.S., rare and endangered plant species are those that have come under federal protection with the Endangered Species Act (ESA) of 1973 [2]. Scientists assess the population sizes, distributions, and trends of the plants in a region. If any species has few individuals, is limited to a few sites, and shows a trend of population decrease, then it is proposed as a candidate for "listing" (placing on the Endangered Species List). The candidates are carefully reviewed before they become officially listed species. Once on the list, the species is offered some federal protection. The ESA statue includes two key provisions: the species must be saved from extinction and there must be a plan for its recovery so that it is no longer in danger of extinction. This second provision, that of recovering the species, is aimed at the eventual removal of species from the list.

As of June 2003, there were 715 total U.S. flowering plant species on the ESA list with 144 in the threatened category and 517 endangered [3]. These species occur throughout the U.S, although approximately one third of the listed flowering plant species occur in Hawai'i.

Understanding the habitat of endangered plant species is an obvious key to the maintenance of the existing populations. There are two parts to the habitat surveillance. Observations need to be made on the ESA-listed plants, and the characteristics of the environment in their immediate neighborhood need to be monitored. In addition, if we are to recover the species then we must also know the environmental conditions in the surrounding region. Knowing the larger pattern of environmental conditions should give us some insight into why the current distribution of the species is limited. It may be, for example, that the rainfall is significantly different in the surrounding area and this limits the reproductive success of individuals in the drier areas or provides significant benefits to a competitive species.

The following section describes a system that meets the general requires of monitoring rare and endangered plants and their environments.

54.2 The Monitoring System

Any monitoring system that involves federally protected rare and endangered plants must not put the population in any further danger. While this clearly means that no destructive sampling can be done, it also prohibits changes to the local environment that might also harm the plants. This constraint sets some broad limitations on instrumentation.

Plant monitoring equipment must not have a physical effect on the plants. This includes shading the plants, modifying the soil conditions, intercepting rainfall, or altering the wind pattern. In part, these are equipment size and proximity constraints. In addition, it is important that the monitoring equipment should not call attention to the plants. This implies that equipment should be as small as possible and, if possible, be able to be hidden.

To the extent possible, the plants and the environment should be monitored remotely. Visits to areas with ESA-listed plant species can negatively impact the environment (such as by soil compaction or transporting alien seeds into the area). As a result, the monitoring equipment should be designed to be highly reliable, be able to survive field use for extended periods (at least several years), and require, at most, infrequent servicing (such as battery changes).

Traditional field weather stations are large, often with rainfall and temperature sensors standing about 2 m tall and wind sensors on a mast. Recordings of weather information are either periodically transferred in the field or the unit may be equipped with data transmission capabilities. Most of the weather stations that are used for long-term measurements are sufficiently close to habitation that they can be connected to telephone lines for data transfer. Some stations use cellular phone links. While these weather stations provide a key backbone of reliable, high-quality data, they are not well suited to

© 2005 by Chapman & Hall/CRC

the needs of rare plants. It is not just a size constraint. Endangered plant populations are generally not found conveniently located near communication facilities.

Installing equipment that will monitor both the endangered species and their environments obviously requires some physical proximity to the plants. At the same time, the equipment needs to be noninvasive. Two general and broadly complementary approaches have been used to meet these requirements. One strategy is to make all of the equipment as small as possible. The other is to hide the equipment. Both of these approaches have implications about the types of data that are collected. For example, standardized rainfall sensors (see below) have a 6 in (15.4 cm) diameter collecting funnel. This is hard to disguise. While a smaller diameter funnel would be possible, it may be better to consider a completely different design that does not attempt to measure rainfall amounts directly near the target plant population. Instead, it may be more appropriate to measure an aspect of rainfall that can be correlated with a standardized measure. This allows the larger equipment to be located at a great enough distance away that its presence does not draw attention to the endangered plants. An example of a surrogate measurement would be rainfall duration. This can be done with a sensor that is both small and quite unobtrusive, such as two parallel conductors that will be shorted together when wet.

The point that we would like to emphasize is that the environmental monitoring system does not have to be identical to traditional designs. A system that is built as a network of sensors provides many new opportunities for fundamentally different approaches.

Visual reconnaissance of the plants allows the collection of important data. Similar care must be taken in the design of the sensors to make sure that there is enough resolution to capture significant life-history events. For example, close-up images might be required to see the initiation of flowering. At the same time, these sensors should not be so close that variability within the plant is missed, or other important events are not seen.

Our experience with images that document a plant's life history events has emphasized the value of periodic high-resolution still images over video recordings. This is not just a matter of data collection frequency. Still cameras generally have image sensors with a larger dynamic range and which possess better optical properties than video systems. This means that you are more likely to be able to see the details needed. Video is important for monitoring animals, but most plant phenological events can be satisfactorily captured by a time series of still images.

Images also have considerable value when trying to interpret the measurements of the other environmental conditions. Seeing the structure of the clouds in a picture helps improve the understanding of the solar radiation measurements, for example.

Having a near-real-time system is very important. There are some situations that will probably require on-site follow up to understand the full implications of a particular event. Remotely monitoring the field conditions, particularly during periods with critical weather, should provide enough information to decide when to make a trip to the study site. An example is heavy rain during a critical event such as seeding development. An on-site investigation, if it is timed right, will likely reveal the actual impact of the rainfall in ways that would be impractical to instrument fully. This example emphasizes that an important goal of the monitoring is to make sure that all field visits are timed for maximum effectiveness while minimizing routine activities around the plants.

The nature of most plant life histories calls for a monitoring system that will operate for several years. This means that renewable energy sources, such as solar panels, will probably be used. Alternatively, the system must operate on extremely limited power. This adds to the challenges in designing a system that will meet the constraints of use around endangered plant species.

54.3 Typical Studies Involving Plant Monitoring

There are many applications of wireless systems of sensors in plant monitoring. The description above has focused on the application to rare and endangered species. The generality of this system can be seen in other monitoring situations.

© 2005 by Chapman & Hall/CRC

Crop monitoring, and the use of these data in models, is becoming a sophisticated agricultural management tool. There are many facets to such monitoring. These involve the use of different measurement scales (from satellite-based remote sensing to in-field sensor systems) and a range of sensors (from traditional weather instrumentation to multi-spectral systems). Even relatively simple systems, such as using NOAA air temperature data to calculate the day-degrees (the number of days with temperatures above some threshold temperature), have allowed predictions of when to harvest crops that have been used for many years. Crop models have become much more sophisticated, however, and can now be used to make a variety of predictions so that farmers can be more astute managers. Crop performance over large areas can be estimated [4]. Most of these agriculturally related systems monitor changes on a daily or weekly basis within a growing season.

At the other end of the temporal scale are those studies that monitor the occurrence of phenological events to help understand changes such as global warming. Plants (and animals) are often sensitive indicators of subtle environmental changes. The small temperature changes of the past century have already been seen in changes in more than 80% of the 143 studies of plants and animals analyzed by Root *et al.* [5].

There are many situations where plants need to be monitored so that animal interaction events can be recorded. The types of event include pollination and herbivory. These may happen very infrequently and over a brief period. This contrasts with monitoring that involves measuring slow but relatively steady changes, such as the growth of an individual. If the animal can be detected, such as with sound or passive infrared, then the sensors can begin more intensive monitoring and image capture.

In summary, it is obvious that there are many types of system that need to be monitored. The requirements differ based on the goal of the monitoring.

54.4 Sensor Networks

The emphasis of many systems of environmental measurement has focused on the temporal changes in the major climate factors, such as temperature and rainfall. While temporal patterns are obviously very important, it is likely that the spatial patterning of the environmental is equally important. The cost of placing many traditional sensors on a site, maintaining these sensors, and interpreting the data has been prohibitive except in a few well-funded studies. New designs of networked microsensors reporting on a near-real-time basis offer a promising alternative. The implementation of such a system involves a number of considerations that require careful planning.

The layout of a sensor network that investigates environments, especially those surrounding rare plants species, should be of a size and arrangement that will detect gradients, if they are present. For example, areas with strong topographic relief are very likely to have rainfall gradients, and if the elevation change is great enough, then substantial temperature gradients as well. Discovering the gradient pattern and its magnitude is important, since such microclimatic differences between the habitat in which a plant is growing and where it is absent may explain this distribution pattern. Therefore, the overall layout should be designed with careful attention to the hypothesized environmental patterns, as well as the general characteristics of the species being studied.

In some cases, the layout of the sensors may be needed to observe phenomena whose location is not known or is not easily predicted. An example, relative to rare plants, is the need to monitor herbivores that may be eating the plants. In many such cases it is not clear ahead of time which species is a likely consumer or where they can be observed. It may take several modifications of the sensor layout before basic information is known. At that point, it may be possible to adopt a different sensor layout that examines the herbivory process in detail.

It has been mentioned before, but is worth repeating, that the general location of the sensors, and the supporting ancillary equipment, should avoid changing the local environment, especially in the vicinity of endangered species. The goal is to have a sensor system that improves access to what are otherwise remote (and perhaps fragile) areas. The overall system should have good long-term unattended

© 2005 by Chapman & Hall/CRC

operational capabilities. This should include appropriate redundancy in sensors, power, and networking components.

The connection of the sensor network to the Internet, or otherwise retrieving data to an attended base location, allows the near-real-time monitoring of the field site. Designing sensors and data analysis systems that function to alert researchers promotes the concept of limiting field visits to those times when critical events are occurring that will benefit from human observation. A variety of extreme events qualify as triggers for on-site follow-up visits, including intense rainfall, flooding, prolonged drought, or intense winds. The system should also alert researchers when there has been a catastrophic failure of the system, so that it can be repaired with minimal delay.

The sensor system does not need to consist of identical units. A system that has a variety of sensors, such as those that collect both rainfall amounts and wetness events (the periods with either precipitation or fog and clouds), is likely to improve the resolution of environmental information. A few rainfall-collecting sensors, which are large and hard to disguise, can be used in areas where their presence does not interfere with the plants. These can be enhanced, and to a certain extent correlated, with smaller moisture detectors that are located both near the collecting sensors and the plants. The combination of the two types of sensor is likely to give much more information about the amount and pattern of the moisture over the area being studied than if a single type of sensor is used. The important point is that some ''nontraditional'' sensors, especially when they are combined with traditional sensors in an appropriately designed network, are likely to provide a richer set of environmental information than has been available to researchers studying rare plant distributions.

54.5 Data Characteristics and Sensor Requirements

54.5.1 Weather Data

Air temperature, relative humidity, barometric (air) pressure, rainfall amount, wind speed, and wind direction are standard measurements taken by weather monitoring stations. The air pressure measurements are generally not used in plant studies. In addition, solar radiation is a very useful measurement that should be included if possible.

Digital sensors are readily available for all basic weather parameters (e.g. Onset Corp., Dallas Semiconductor). Humidity measurements generally use a thin-film sensor, and temperature is measured with a thermocouple sensor. Some care is needed to make sure that these sensors are in proper enclosures, i.e. shaded from direct sunlight and precipitation and with ample air circulation.

Rainfall amounts are accumulated using a tipping-bucket sensor. These event sensors generally record each 0.01 in of rainfall. Rainfall is collected in a funnel, generally 6 in (15.4 cm) in diameter. Hourly reporting is a standard measurement interval. Reporting is generally adjusted to report starting on the hour. If daily reporting is done, then the accumulation is reset at midnight.

Wind speeds are measured in a variety of ways, all of which provide an instantaneous measurement value. These may be reported as an average, sometimes with a gust (1 min peak) value. Propeller devices have a minimal threshold below which they cannot measure the wind speed, often around 1 m/s. Wind direction, also determined as an instantaneous value, is generally reported as a compass direction. See Webmet [6] for information on computing the vector mean wind speed and direction.

Solar radiation data are much less commonly reported. The radiation characteristics measured by the sensors vary considerably. Simple light measurements provide a very coarse value and may be adequate for general survey considerations. Critical measurements may require a photosynthetic light (PAR) sensor that closely matches the energy spectrum acquired by typical flowering plants.

54.5.2 Soil Data

Soil conditions, such as soil moisture, are often critical to the growth and survival of plants. Digital sensors for soil moisture are now available (Onset Corp). These are relatively temperature and salinity

© 2005 by Chapman & Hall/CRC

insensitive. They read the volume water content in the range from 0 to 40.5% with an accuracy of approximately ±3%. Soil temperature sensors are similar or identical to air temperature sensors.

Both soil temperature and soil moisture may vary substantially over short distances and depend on the soil composition, slope, type of vegetation, and other factors. This suggests that sensors should be placed at several soil depths and in different locations.

54.5.3 Images

Periodic pictures of the site being monitored are very useful if they have sufficient resolution and dynamic range. Still images, such as those produced with two-megapixel (or greater) digital cameras, meet these standards better than video images. In general, images should be timed to correspond to the collection of weather data (e.g. hourly). Color images, while not essential, may provide essential information such as differentiating between clear and cloudy sky conditions or helping to see the presence of flowers on a plant. Image collection has not been a standard part of the data collection protocol for plant monitoring. Our experience has shown that it can be particularly valuable if high-quality images are collected at consistent monitoring intervals over long periods of time.

54.5.4 Event Detection

There is a broad range of events that are of interest for plant monitoring. Many of these have a low probability of occurrence, but they can have a dramatic (perhaps catastrophic) impact. Examples include fires, lightning, and floods. While lightning sensors are readily available, monitoring the occurrence of the other events requires the adaptation of other sensors. Additional important events that are not as closely associated with specialized sensors, such as grazing activity or pollination, may require analyses of images to determine their occurrence. Intrusion detection is a likely candidate to trigger image analysis; however, the sensor requirements must be established relative to specific targets. Large grazing mammals present a qualitatively different problem than a pollinating insect.

54.6 Spatial and Temporal Scales: Different Monitoring Requirements

The precision of any particular sensor requires detailed analysis before it is selected for incorporation in any plant monitoring system. The basic issue is whether it is better to support fewer higher precision sensors or more that are lower precision. Researchers have traditionally used high-resolution sensors. The costs may be so great that monitoring is limited to a single set of sensors (e.g. one weather station). The benefit of such a system is that the accuracy allows its measurements to be compared with other similar systems in other areas. If there is a local gradient in the environment, however, then a limited number of high-precision sensors may not provide enough spatial coverage to measure the trend. As a result, the environmental factors limiting the plant distribution may not be detected.

The sensor accuracies generally used with the common environmental measurements are:

- Temperature, ±1–2°F
- Humidity, ±3%
- Wind speed, ±1 to 3%
- Rainfall amount, ±5%

There are a number of ways to measure solar radiation. Examples of differences in sensor costs can be seen by comparing light measurement with a diode (at approximately $2 per sensor) and PAR sensors (at approximately $175 per sensor).

Medium-precision systems, especially when they are widely deployed, appear to be well matched to the needs of monitoring heterogeneous environments. Design considerations should examine the use

© 2005 by Chapman & Hall/CRC

of low precision but very numerous sensors. It is likely that a network using such sensors holds some potential for efficiently uncovering some types of spatial patterns and temporal trends.

54.7 Network Characteristics

Sensors can be networked for a variety of purposes, the most common being for sending data from the field to a base station. Other goals might include coordination among sensors for event detection or computation within the network.

Some commercial weather stations transmit data from the weather station to a base-station receiver using 900 Mhz spread-spectrum modems. With appropriate antennas, line-of-sight distances of over 30 miles have been reported [7]. Telephone lines are used if they are available. Other alternatives include radio data links or cellular telephone connections.

For weather measurements, little data are sent; so, unless a very large number or a high frequency of measurements must be transmitted, all these technologies are suitable. Other applications, including periodic high-resolution images, require higher data rates, though even a 700 Kbyte JPEG image (typical from a two-megapixel camera) once an hour only requires about 1600 bits/s on average. In comparison, a weather station transmitting 120 bytes once a minute only requires 2 bits/s, and 100 such weather stations still only require about 200 bits/s.

Traditional telephony and most cellular telephones are bandlimited to (before compression) about 64 Kb/s and 9.6 Kb/s respectively. Newer cellular technologies allow data rates in the megabit/second range. Satellite technology is capable of carrying large data rates, but the cost per bit may be high, as is currently the case for cellular technology.

Radio data links vary from lows of around 9.6 Kb/s (many serial radios) to highs of 11 Mb/s (802.11b) and 56 Mb/s (the less common 802.11a). Radios provide low cost per bit, since the costs are related only to purchasing the hardware and providing electrical power.

All radio technologies have a distance that is variable depending on the antenna and the power level used. Power levels may be limited by the hardware, often to obey regulations. Antennas may also be a given for a given hardware (or limited by regulation), or may be selectable. In general, an antenna provides a gain by focusing data in a given direction. Omnidirectional antennas distribute the data 360° within a plane perpendicular to the axis of the antenna. The signal is strong within a number of degrees of the plane, e.g. 30° or 20°. The more the signal is focused near the plane, the greater the gain of the antenna, and the less power can be received away from the plane. Directional antennas instead focus the signal in a cone, with most of the signal strength within a certain angle from the axis of the cone. Again, the smaller the angle the greater the gain. Most directional antennas have higher gain than most omnidirectional antennas, but omnidirectional antennas used for communication on a plane (e.g. on the Earth's surface) have no need to be aimed.

Antenna placement also affects range. When an antenna is near a conducting surface, such as the surface of the Earth, the power falls off very rapidly with distance, typically as r^4. This is due to the electromagnetic wave being reflected by the conductor, which typically leads to near-cancellation of the wave. The signal for antennas that are far from any conductor, on the other hand, tends to only fall off as $1/r^2$. The actual attenuation of the signal with distance depends on a number of factors, including the directionality of the antenna and the overall geometry of the configuration of sender, receiver, and reflective surface.

With typical antennas and power levels, most radio modems will work over a distance of at most a few hundred meters, with 802.11 being similar but perhaps somewhat higher. Bluetooth, a relatively low-power radio technology, is designed for a communication range of 10 m, though part of the standard (Class C) allows for communication up to 100 m. Cellular communication can, in theory, extend several miles from a cell phone tower, but cellular can support more bandwidth and transmissions if the cells are small, so current cellular systems tend to keep cells as small as possible. The range of satellite systems is limited to the area visible from the satellite itself, but this may be very large,

© 2005 by Chapman & Hall/CRC

since some satellites have a footprint covering most of the continental U.S. In addition, many modern systems provide a number of satellites that can cover the entire planet.

With carefully aimed directional antennas and line-of-sight conditions, the same radio technologies can reach across many kilometers, though weather, including fog, clouds, and precipitation, can interfere with such transmissions.

The sensitivity to weather varies with the frequency of the radio signal. The 2.4 GHz microwave band, used in microwave ovens as well as 802.11, 802.11b, and Bluetooth, is among the most affected, though most radio frequencies in common use will be affected by weather and vegetation (which contains water). Other interference can also affect radio transmission. Technologies which use spread-spectrum distribute the signal across different channels and are thus able to avoid more sources of interference than conventional single-channel technologies that do not use spread-spectrum.

54.8 Deployment Issues

The deployment of sensors in a wireless sensor network used to monitor plants is affected by many factors, including accessibility (e.g. placing sensors or radios high in trees or in remote mountain locations), radio connectivity, and coverage. Other factors may include the ability to conceal the sensor or to take pictures of specific plants. Some deployments may need to be made in dense vegetation. Visits to proposed deployment sites early in the planning process are essential.

The arrangement of the nodes is important. Researchers often want to control the precise location of the nodes. For example, it may appear to be conceptually useful to have a regular spacing among nodes, such as arranging them in a regular grid. The actual details of the site, however, will impose many constraints on locating nodes, especially if they need to be concealed. It is better to have a general plan and to make sure that there is some flexibility in its implementation. This calls for considerably more understanding of the communication properties of the nodes in the actual deployment conditions.

In a wireless *ad hoc* sensor network, each unit relays the data collected by other units as well as generating its own data. In such a network, and if the units are not guaranteed to be 100% reliable, it may be desirable to place additional units to maintain radio connectivity should a few of the nodes fail.

The number of nodes used in the monitoring system is also a critical issue. In extreme cases, most of the power used by the network as a whole is used to transmit data. In such cases, and if the radio range falls off as $1/r^4$, as is usually the case, the power consumption is minimized by having the largest possible number of nodes with the shortest possible radio range. Doing this may also optimize the overall bandwidth (bits/second) of the network as a whole, though the benefits of this depend critically on the overall communication pattern. For example, if all the data are sent to a base station then there is no benefit, as the bandwidth of the base station forms the bottleneck for the entire network.

Since minimizing the power may not minimize the cost, careful forethought is needed both at the radio selection stage and when planning the deployment. Cost is also usually divided into design cost, which is amortized when more of the units are built, and a per-unit (materials and assembly) cost, which is greater when more units are built.

One of the biggest challenges is to hide the instruments. Small items are relatively easy to conceal, but large components, such as solar panels, wind and rain sensors, and cameras, present some difficulties.

Long-term deployments require that all the instruments be adequately protected from the environment. Small openings seem to invite water, for example. Unprotected connectors may quickly corrode. The opposite type of protection is also important, as it is critical to keep the instrumentation from affecting the local environment. For example, corrosion or battery leaks could introduce toxins in the environment that could be detrimental to the organisms being studied.

54.9 Data Utilization

Making effective use of the data, once they have been collected, can be challenging. Typical situations range from those that are highly determined, with a fixed set of questions to answer, to those that are part of an open-ended investigation.

© 2005 by Chapman & Hall/CRC

If the goals of the data collection are known in advance, then it is often possible to perform much of the necessary computation on the network nodes, decreasing the amount of data that must be transmitted or the distance over which the data are transmitted. In general, such goals may consist of collecting appropriate statistics and detecting specific events. Statistics (e.g. minimum, maximum, and average temperature) can often be computed in a distributed fashion so that data transmission is minimized. Event detection covers a broader range of computations, and may or may not be suitable for distributed implementation. Ultimately, nodes may only transmit once events are detected, potentially greatly reducing the power consumption of the network.

A completely different approach to data collection is to leave the data stored on the nodes (forming a distributed database) and allow queries to be performed on this distributed database. Some approaches combine queries with event detection, such as diffusion [8], or queries alone can be used. In such cases, the data are delivered on-demand, potentially substantially minimizing the amount of data transmission.

Once the data have been collected, they must be put to use. Typically, the amounts of data are such that looking directly at the numbers has limited usefulness. Instead, most users prefer to use algorithms to visualize the data, e.g. by graphing temporal changes in the data or creating maps to display many data points at once. As for event detection, visualization is easiest when the goals of the data collection are known exactly. For example, a farmer wishing to know whether a sprinkler system has delivered the expected amount of water may study a map of current soil moisture, and perhaps trend different areas over different time periods. Scientists studying endangered plant species and trying to figure out why the species are threatened, on the other hand, may need to visualize the data in many different ways to identify cause-and-effect relationships that may be affecting the plants and their ecosystem. Even evaluating the health of an ecosystem may be somewhat challenging, if there is an absence of data defining what is normal and healthy.

References

[1] Leith, H. (ed.), *Phenology and Seasonality Modeling*, Springer-Verlag, New York, 1974.
[2] U.S. Fish and Wildlife Service, http://endangered.fws.gov/esa.html, 2001 (last accessed on 8/18/2004).
[3] U.S. Fish and Wildlife Service, http://ecos.fws.gov/tess_public/html/boxscore.html, 2003 (last accessed on 8/18/2004).
[4] EARS (Environmental Analysis and Remote Sensing), http://www.earlywarning.nl/earlywarning/index.htm, 2003 (last accessed on 8/18/2004).
[5] Root, T.L. *et al.*, Fingerprints of global warming on animals and plants, *Nature* 421, 57, 2003.
[6] Webmet, http://www.webmet.com/met_monitoring162.html (last accessed on 8/18/2004).
[7] Weathershop, http://www.weathershop.com/WWN_rangetest.htm, 2003 (last accessed on 8/18/2004).
[8] Heidemann, J. *et al.*, Building efficient wireless sensor networks with low-level naming, in *Proceedings of the Symposium on Operating Systems Principles*, Chateau Lake Louise, Banff, Alberta, Canada, 2001, 146, http://www.isi.edu/~johnh/PAPERS/Heidemann01c.html.

© 2005 by Chapman & Hall/CRC

55

Designing Distributed Sensor Applications for Wireless Mesh Networks

Robert Poor and Cliff Bowman

55.1 Introduction

More than 2000 years before Eckert and Mauchly conceived of the logic and electronics that would become ENIAC, Plato penned the words, *Necessity is the mother of invention*. This adage has particular relevance in the rapid growth of wireless mesh networks, where commercial, industrial, and military applications have spurred innovation and fostered technological advances. A broadening array of practical solutions to industry challenges now rely on distributed sensors linked wirelessly with networks based on mesh topologies.

Real-world applications based on wireless mesh networks and distributed sensors take a wide variety of forms. Imagine high-rise buildings in earthquake-prone southern California with embedded strain gauges contained in the structural members, delivering data wirelessly to monitor the integrity of the structure during seismic events. Freight shipments in ships or trucks are monitored for temperature, shock, or vibration using wireless sensors in the cargo area that store data en route and deliver it upon docking. Petroleum pumping stations in frigid regions maintain oil flow at a precise degree of viscosity using embedded sensors in the pipelines linked to a feedback mechanism that controls individual heaters. Environmental monitors in orchards and vineyards guide irrigation schedules and fertilization, and provide alerts if frost danger becomes evident. Water treatment facilities use wireless sensors to monitor turbidity levels at the final critical stages of treatment and issue warnings if the monitored values exceed limits.

The progression towards wireless sensor networks is inevitable. The decreasing costs and increasing sophistication of silicon-based integrated circuits, has led to low-power processors, specialized chipsets, and inexpensive wireless components, encouraging broader acceptance of the technology. The evolution of the Internet offers an example of a clear progression, moving from one connection for many people (the mainframe mode) to a single connection for each person (the microcomputer/laptop model).

© 2005 by Chapman & Hall/CRC

The next stage in this progression, providing many connections per person, encompasses the sensor network model. Within this model, collections of sensors deliver data to a person through multiple connections.

As sensor networks become more ubiquitous, the sheer volume of deployed sensors makes it essential that these networks are designed to be self-maintaining and self-healing. Already the number of sensor devices exceeds the population of the planet, and more than 7.5 billion new devices are manufactured every year. Developers who want to capitalize on the benefits of this technology must realize that sensor networks have distinct differences from conventional wired and wireless networks. Workable designs favor simplified deployment, low power consumption, and reliable, unattended operation.

Recent advances in wireless mesh network technologies open significant new opportunities for developers. The characteristic properties of mesh networks fit many types of embedded application, where resources — such as power, memory, and processing capabilities — are constricted. With easy deployment and self-healing capabilities, mesh networks satisfy the primary requirements of well-designed sensor networks. Wireless mesh systems can be built using inexpensive, commonly available eight-bit processors.

This chapter describes the principles underlying application development for wireless mesh networks and provides several examples of real-world applications that benefit from this technology. Getting optimal performance from a wireless mesh network typically requires a fresh design approach. A straight translation of an existing wired network application to a wireless mesh implementation often yields disappointing results. Following a comparison of the popular network topologies, this chapter presents guidelines and design principles that lead to successful deployments of distributed sensor networks using wireless mesh systems.

55.2 Characteristics of Mesh Networking Technology

Mesh networking technology owes much of its increasing popularity to the inherent reliability of redundant message paths. This fail-safe approach to communication and control adapts well to implementations in manufacturing, public service utilities, industrial control [1], and military applications [2]. Mesh networking offers a number of distinct benefits, including:

- *Highly scalable network infrastructure*. Each node in a mesh serves as a relay point, resulting in a network infrastructure that grows along with the network. Because of this design framework, mesh networks support incremental installation paths. Initial investments in the technology are also minimized, since a very basic network can be quickly deployed and then extended as required.
- *Simplified deployment in distributed environments*. Deploying a mesh network is typically easier than deploying networks using other topologies, particularly when propagation varies widely over a geographic area or over time. Once deployed, mesh networks can automatically take advantage of "good" variations in propagation [3].
- *Energy efficiency advantages*. Developers working on applications intended for embedded implementations can capitalize on certain characteristics of mesh networking. The success of many battery-powered embedded applications relies on achieving maximum energy efficiency, extending battery life as much as possible. Overall power drain attributable to r^n path loss in wireless mesh architectures tends to be lower because, on average, the value of r is less [4]. This lets developers significantly reduce transmitter power to reduce the corresponding power drain.
- *Minimal processing requirements*. Cost-effective embedded applications must often rely on low-power processors with limited memory. To overcome this challenge, software engineers and developers have constructed loop-free routing algorithms explicitly for mesh networks. These memory- and processor-efficient routing algorithms [5,6] make it possible for developers to implement large-scale networks using modest processors with very low power requirements [7].

© 2005 by Chapman & Hall/CRC

55.2.1 Design Considerations from a Developer's Perspective

Adapting existing applications to successful wireless mesh network implementations often requires re-evaluating fundamental design considerations. The data capacity and the capabilities of available wireless devices are typically more restrictive than the capabilities of an equivalent wired network. Developers evaluating wireless mesh projects should recognize that their existing messaging models do not translate seamlessly to available mesh devices. Throughput and the data capacity of the wireless mesh network become prime considerations, which can require rethinking the architecture of the network.

Upon further investigation, many developers discover a basic truth — the design of many embedded protocols is tightly linked to a wired medium. When developers simply translate an existing legacy application, the performance of the wireless mesh network can be disappointing. Often, the performance results do not reflect limitations of wireless mesh networking, but rather a misuse of the technology.

Although developers must sometimes integrate mesh applications with legacy systems, dropping irrelevant design practices that apply to outdated wired systems can frequently help improve efficiency and network performance. In real-world situations, designers often encounter challenges that fall somewhere between maintaining absolute interoperability with legacy systems and creating a standalone wireless mesh network with distributed sensors as a fresh design. Drawing on the guidelines presented in this chapter, design trade-offs can usually be managed in a reasonable way. Reliability, scalability, adaptability, and efficiency, the hallmarks of wireless mesh networks, represent achievable goals through intelligent engineering. Real-world examples of practical mesh networking implementations appear in Section 55.5.

55.3 Comparison of Popular Network Topologies

Two distinctive properties that help characterize communications networks are:

- *Topology*. Topology refers to the pattern by which a network's nodes are organized. Popular network topologies are bus, star, and mesh. The network topology determines the kinds of connection that are possible between nodes. Essentially, the topology creates a framework that controls how individual network devices communicate.
- *Medium access*. Medium access defines the rules by which an individual node can transmit on the shared communication medium (bus, star, or mesh). These rules can dramatically affect network behavior and performance. A trend that is evident in recent designs is access responsibility distributed among the nodes.

Figure 55.1 illustrates the basic topological structures that differentiate bus, star, and mesh networks. Assume that a message must pass from node A to node F through each of these topologies. In all cases,

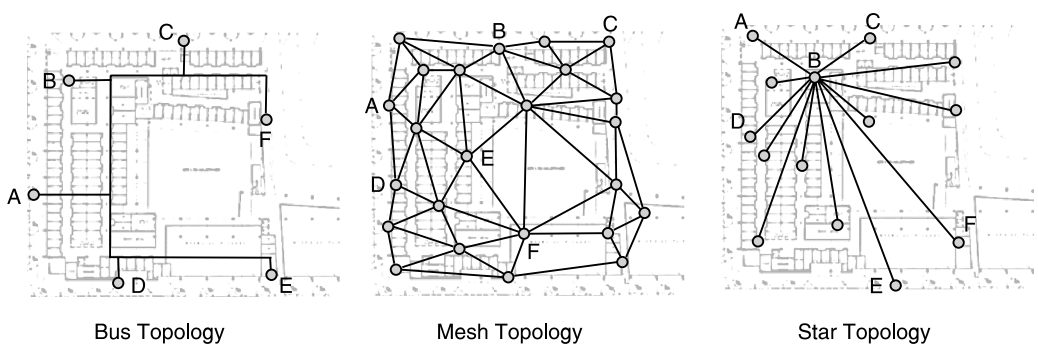


Figure 55.1. Bus, star, and mesh network topologies.

© 2005 by Chapman & Hall/CRC

the organization of the network topology determines the paths by which the message can travel. The mechanisms by which each node gains access to the shared communication path depend on the protocol applied to the selected topology and the medium access techniques in use.

55.3.1 Transferring a Message within a Bus Topology

In the bus topology shown in Figure 55.1, every node can communicate with every other node — the message travels directly from node A to node F. Wired networks that operate in this manner include Ethernet local-area networks (LANs), Profibus, Modbus, and a number of proprietary systems that use the multi-drop RS-485 interface. Wireless networks, in some cases, also operate in a manner similar to a bus. An example of this is when a conference room of 802.11 devices is set to *ad hoc* mode.

Routing on a shared bus, however, is more complex than it appears. If two nodes attempt to transmit on the bus at the same time, then their messages can collide, resulting in garbled information. Employing some form of medium access can minimize the chances of collisions. Some systems, such as Modbus, limit themselves to query/response messaging. For example, in a system employing Modbus, a *master* node owns the bus and a *slave* may transmit only when the *master* sends it a query. Other systems use scheduling schemes, such as the technique used with 802.11 *ad hoc* mode: each node can transmit only during a specified window of time assigned to it. A third strategy, implemented in Ethernet and known as carrier sense multiple access (CSMA), relies on carrier-sense hardware contained in each node. By sensing the state of a signal that indicates the bus is in use, each node can detect whether another node is already transmitting before it attempts to gain access to the bus.

Because messages travel directly from source to destination within a bus network, relay failure is not an issue. The vulnerability of bus systems, however, depends on the effectiveness of their medium access strategies, as well as in the integrity of the bus itself. In a Modbus network, where medium access is controlled exclusively by the master, communications are disrupted if the master node fails. Networks that rely on scheduling, where each node gets a specified window of time in which to transmit, also have a single point of vulnerability: the nodes typically depend on a synchronizing beacon to find their window. If the beacon station is lost, then network recovery can take a significant amount of time. The technique used by Ethernet, based on detection of a signal that indicates the bus is in use, presents less vulnerability. Because access responsibility is distributed among the Ethernet nodes, the failure of a single node does not affect any of the other members of the network.

Bus systems, by their nature, all share a common vulnerability: the potential for losing access to an entire section of the network through bus failure. This can occur if a wired segment of the bus is cut or a wireless segment of the bus is jammed.

55.3.2 Transferring a Message within a Star Topology

Star networks employ a different method of organization. Within a star topology, each transmitted message travels a fixed path: node A can transmit only to the master node B, which then relays the message to node F. If separate cables are used to link each of the satellite nodes to node B, then medium access does not present a problem. In the case of a shared medium, such as wireless, the common technique is to let node B determine which node can transmit. One example of this approach is the medium access scheme used within a Bluetooth piconet.

One inherent vulnerability of the star topology affects its reliability: if the master node fails, all communications on the network become disrupted. In shared-medium systems, such as Bluetooth, the member nodes can select another master and communications can be re-established after a delay. Recovery cannot be initiated, however, in some star topology configurations, such as when the single hub of a wireless LAN fails. In addition, if the path between the master and a node is blocked, that node can no longer participate in the network.

© 2005 by Chapman & Hall/CRC

55.3.3 Transferring a Message within a Mesh Topology

Within a mesh network, messages can travel over multiple paths. A message transferred from node A to node F can be routed from A to B to F or from A to E to F. Many alternate paths can be used as well, and this redundancy is a characteristic that increases the reliability of mesh networks. In a well-connected mesh network, the failure of a single node (node B, for example) only affects communications for that node. Messages previously directed through the failed node can be automatically rerouted.

Link failure, as occurs with the severing of a network cable or the blocking of a radio-frequency (RF) path, has much less effect on a mesh network than on other network topologies. The redundant routes available within a mesh network let traffic navigate around the broken link. This ensures that link failure cannot exclude a node from the network.

The nodes in a wireless mesh network typically use a shared RF channel, requiring some method to arbitrate medium access. The method commonly employed is CSMA. Since the hardware that supports CSMA is often built into each radio, implementing medium access can be fairly simple. As mentioned previously, the distributed strategy used by CSMA protects the network against the failure of a single node.

The medium access strategy that applies to mesh networks is similar to the strategy that applies to Ethernet LANs, with one important difference: wired Ethernet LANs are usually bus networks,[1] so only one node can transmit at a time. In wireless mesh networks, nodes relay messages for each other, allowing the use of low-power transmitters. By reducing power to the point that transmissions reach only nearby nodes, the channel remains available for nodes that are beyond the range of the transmission. This phenomenon, known as *spatial multiplexing*, exists when multiple messages can travel simultaneously in different parts of the network. For example, as shown in Figure 55.1, traffic can pass between node A and node D at the same time that node C and node F exchange individual messages. The use of spatial multiplexing increases the effective data capacity of the network.

55.4 Basic Guidelines for Designing Practical Mesh Networks

Developers and engineers contemplating designs based on mesh networks can optimize their implementations by the following guidelines:

- *Distribute control tasks*. Mesh networks operate more effectively if tasks and messaging operations are distributed, rather than centralized. Centralizing tasks creates a network traffic pattern that focuses on the node controlling the process — messages either originate or terminate at an indicated node. Distributing control to several different points in a mesh network, particularly if these points are geographically separated, causes traffic around each point to flow independently. Multiple messages can be handled simultaneously (using the principle of spatial multiplexing), which effectively multiplies the capacity of the network. Distributing tasks has another benefit: messages do not need to travel as far across the network. By implementing multiple control points, the average distances from message source to destination can be shortened. This also enhances the overall reliability of the system. If a system relies on a single control point, then the entire system gets shut down if that point fails. In a distributed system, however, even if individual components malfunction, the overall system can often continue to operate. Distributing tasks in this manner contributes to an increase in the long-term reliability of the system.
- *Use exception-based messaging to push the data*. To minimize network traffic and increase efficiency, rely on exception-based messaging to obtain data from nodes. Other techniques for

[1]LAN subnets are often wired as "Star Bus" architectures. Physically, they are star networks, but the hubs are not really nodes — they merely repeat messages onto every arm of the star. Logically, the network functions as a bus.

© 2005 by Chapman & Hall/CRC

exchanging messages, such as polling, generate a significant amount of superfluous network traffic. Exception-based messaging reduces network traffic in two ways: it eliminates the query initiating an exchange and it reduces the number of exchanges to those that indicate a noteworthy change in condition.

- *Avoid query-response messages; let the network work.* Messaging techniques that depend on query/response methods or token passing reduce the efficiency of a mesh network. Traditional non-CSMA messaging models that perform well on earlier-generation network architectures may need to be adapted to mesh networks. Embedded protocols targeted for bus architectures typically rely on message-intensive models tied to query and response patterns or token passing to arbitrate access to the bus. The distributed nature of mesh networks favors the CSMA approach for efficient communication, and the medium access strategies that apply to other technologies only add unnecessary overhead to a mesh network design.
- *Use local control and global monitoring.* Let the sensors and actuators communicate directly. The highest efficiency can be achieved in a mesh network by the distribution of tasks to lower level devices in the network. For example, the control logic for operating an actuator can be embedded within the sensor and used to perform tasks as defined by the application. For binary or limited-state actuators, this can involve simply incorporating a table that specifies the threshold values. Decision making that takes place through programmable logic controllers (PLCs) can be implemented through the individual sensors distributed throughout the network. Localized logic operations minimize reliance on a centralized processor. By reducing unnecessary processor communication, message transfers across the network can be minimized, thus improving overall efficiency.

55.4.1 Parameters for a Typical Mesh Network

Table 55.1 lists the typical characteristics of a commercially available mesh-networking suite based on the emerging IEEE 802.15.4 standard. Some of the other mesh networking technologies being developed for commercial use provide much higher network throughput; the data in the table, however, offer typical values for cost-sensitive embedded applications, such as condition monitoring and building automation.

As indicated in Table 55.1, the ''sustained network capacity'' represents a fraction of the ''channel rate.'' Several factors influence this situation. A half-duplex ''store-and-forward'' strategy reduces the rate by a minimum of 67%, CSMA introduces delays for ''back-off'' timing, and GRAd routing relies on a distance-based delay to select efficient pathways. The precise ratio between network capacity and channel rate in mesh networks will always depend on the nature of the implementation. However, factors similar to those mentioned are probably universal, suggesting that the ratio between capacity and channel rate will always be small. In the design of mesh-based solutions, these factors influence the strategy employed and make it necessary for the developer to be continuously aware of the available network capacity.

55.5 Examples of Practical Mesh Network Applications

The following examples illustrate a number of the principles discussed in the previous sections by highlighting design considerations in practical wireless mesh network applications. These examples are

Table 55.1. Typical characteristics of a wireless mesh network

Radio/MAC	IEEE 802.15.4 (CSMA)
Frequency band	2.4 GHz
Power output	+10 dBm
Routing	Multi-hop GRAd
Relaying strategy	Store and Forward
Channel rate	250 Kbps
Sustained network capacity [8]	40 Kbps

© 2005 by Chapman & Hall/CRC

based on actual deployments, but the company names have been changed. In real-world deployments, mesh networks often require a hybrid approach that may include integrating components with existing legacy systems or combining wired and wireless network segments to achieve a design goal. Each example illustrates a particular type of challenge faced by developers in the field.

55.5.1 Equipping a Water Treatment Plant with Distributed Sensors

Indigo Waterworks provides water treatment services for a mid-sized community located in the Rocky Mountains. In an effort to simplify maintenance and reduce costs, senior management at the facility instituted a pilot study to determine whether wireless sensors could be used to transfer data to the central control room. The water treatment plant contained several potential RF interference sources, and staff members expressed concern that the environment would prove unsuitable for wireless applications. Sources of potential interference included the absorption effects of the water on the 2.4 GHz radio signals, the large amounts of iron piping running through the facility, the rebar contained in the thick concrete walls, and the electrical fields given off by the pump motors and switching gear. In this type of environment, there was an immediate concern that RF signals could not be transmitted reliably from sensors to the control room, and that the degree of difficulty in deploying the wireless network might make the entire project impractical.

The sensors used in this application measure the turbidity of the treated water at one of the final stages of the treatment process. The existing wired network in this facility spanned three floors and relayed data to a control room where specialists monitored the effectiveness of the treatment processes. Thick concrete walls, a winding stairwell, and several dozen feet separated the sensors from the control room. These factors complicated the situation for a wireless deployment.

The challenge in this example involved designing and implementing a parallel data delivery system to route information from the turbidity sensors to a mock control room situated beside the actual control room. The instruments were located three floors down, down a stairwell, and on either side of the stairwell. On one side, four sensors occupied a small pipe gallery built as part of the original facility. A later expansion added a larger pipe gallery on the opposite side; this gallery contained eight additional sensors. The deployment consisted of these 12 sensor nodes and additional relays to connect them with the mock control room.

55.5.1.1 Deployment Strategy and Implementation

The management at Indigo Waterworks wanted to answer a number of questions through this test deployment:

- How much time and effort would be required to deploy the wireless network?
- Would the installation and deployment require any special-purpose tools or additional equipment?
- Were specialized skillsets required for any personnel involved in the deployment?
- How reliable would the wireless network be, given the many possible sources of RF interference?
- Is a wireless network practical for the kinds of critical operation performed in a water treatment plant?
- Would the wireless system integrate effectively with the existing Modbus devices used at the facility?

For this environment, the sensor placement and wireless network communication links were organized as shown in Figure 55.2.

After a site evaluation that mapped the positioning of each of the functioning turbidity sensors, the wireless mesh design team placed simulated instruments next to each of the real instruments and deployed the network nodes. Then they installed the relay chain and linked it to the mock control room. Once the wireless network was up and running, information began coming in from each of the sensors, but the reliability from the original pipe gallery was not as high as expected.

© 2005 by Chapman & Hall/CRC

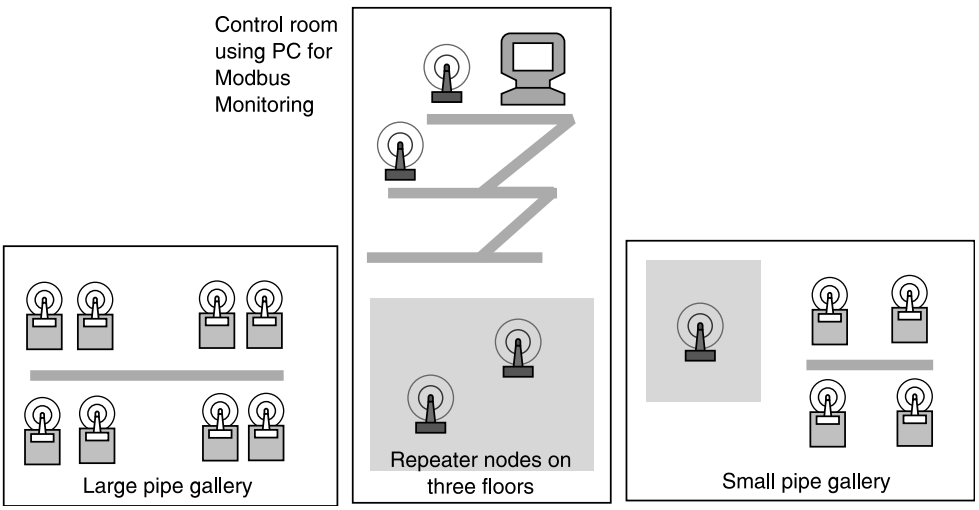


Figure 55.2. Deployment of wireless sensors in relation to the control room.

A visualization tool that provides a complete evaluation of the network indicated that most of the connections and links were functioning normally, but that links were relatively weak. By employing a set of additional repeaters, the team managed to circumvent a significant barrier: a wall of reinforced concrete that was 18 in thick. Using a hole cut for an air duct, the team deployed wireless nodes on either side of the wall near the duct. This was all that was necessary to boost the signal strength sufficiently to bring up the reliability figures.

The connectivity at this point was significantly better. The design team began collecting data and performed a complete tabulation every 24 h to check the reliability of the connections.

55.5.1.2 The Results

During a 4-day interval, the wireless network and sensors functioned at a level of four nines reliability, which indicates that better than 99.99% of the reports were coming back and being successfully logged. The entire deployment, both the initial placing of the nodes and then the re-evaluation and placement of the additional repeater nodes, was completed within 3 h.

The Indigo Waterworks staff members were pleased by both the success rate of the message transfers and the fact that the RF interference proved to be less of an impediment than originally thought. Their instruments, which used the Modbus protocol, did not require any modification to work within the wireless environment. The design team had effectively encapsulated the Modbus packets. Devices throughout the network communicated without awareness that the connectivity involved wireless links. The deployment essentially provided a drop-in replacement for the wired network.

As encouraging as these results were, the nature of the deployment relied on polling techniques, a legacy requirement from the Modbus protocol, to acquire the sensor data. While this process worked very effectively for this particular implementation, the solution does not support the full scalability that can be achieved by a wireless mesh network that uses exception-based processing. Through exception-based processing, the approach could have been re-engineered so that the sensors only delivered data if the turbidity exceeded defined parameters. This type of re-engineering often requires balancing the efficiencies of pure wireless mesh design with the practicalities of a legacy protocol (in this case, Modbus). With intelligent design, engineers need not accommodate the requirements of an earlier protocol.

While this example illustrates the viability of wireless networking within a difficult RF environment, the design guidelines described earlier are not followed. The wireless networking essentially provides a drop-in replacement for existing equipment and helps reduce the costs associated with constructing

© 2005 by Chapman & Hall/CRC

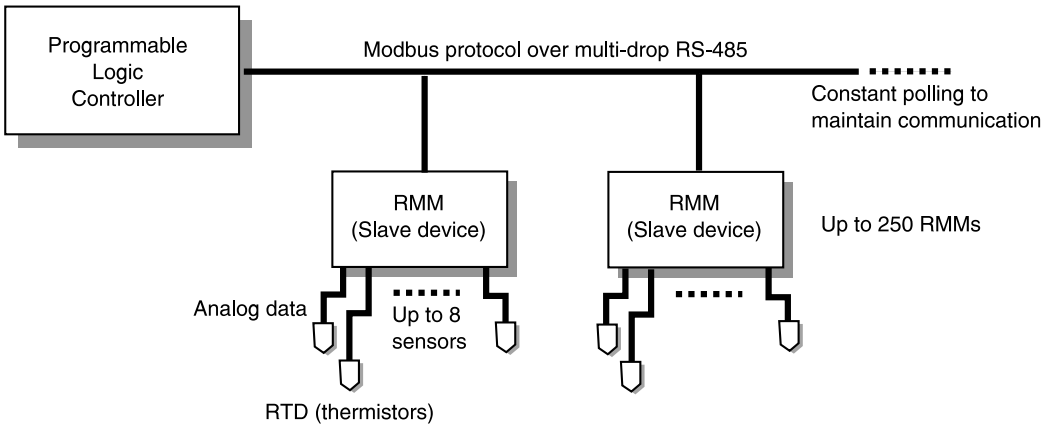


Figure 55.3. Original configuration using Modbus querying.

cable conduits and pulling network cable throughout the facility. Since the nature of many water treatment plants dictates that they are built small and then extended as the needs of the surrounding community increase, the typical approach is to expand the existing plant rather than build a second facility. If the scalability limits were not exceeded, then the wireless network used in this example could be employed to handle the sensor monitoring at a water treatment facility effectively.

55.5.2 Designing a Process Control System Using a Wireless Mesh Network

BlackGold Inc. operates a petrochemical extraction facility in northern Alaska, pumping oil from the ground and heating it to a particular temperature to maintain desired viscosity. An existing system within one of their facilities used distributed sensors communicating by means of the Modbus protocol.

Temperature monitors are installed at several different points in the piping. The technique originally employed was to generate a series of Modbus queries to each instrument sensor in a round-robin fashion. New queries were generated as quickly as the instruments reported back and the results were fed to the controller, which turns heaters on and off for different sections of the pipe. Figure 55.3 illustrates the original system configuration.

55.5.2.1 Deployment Strategy and Implementation

A wireless mesh design team brought in to try to improve the process had to work out a solution that minimized the impact on the existing instrumentation. The team tackled the problem by starting at the data collection point and installing a wireless node onto it. The node simulated the entire network. It answered queries from the controller as quickly as the controller requested information. The link between the central controller and the wireless unit relied on the Modbus, but the query rate was too fast for the network to handle. To resolve the problem, the team engineered the solution so that the wireless node received information from each of the temperature sensors using an exception-based strategy. Any time that a temperature changed or a certain time window was exceeded, the temperature sensor generated a report. This information could then be cached and provided to the primary controller whenever the controller requested it. If the temperature changed, then the node was set up to generate a report of that change immediately. This technique ensured that the wired controller would always have current information on the temperature status of every section of the pipe.

The 1 min timeout interval ensured that the system could detect a failure at one of the temperature sensors. If the sensor failed to report, then the wireless node connected to the sensor would recognize a potential problem and attempt to contact that node. If the contact failed, then the wireless node would report an error back to the primary controller. In this example, the solution was over-engineered, in that the polling took place at more frequent intervals than required by the application. The time

© 2005 by Chapman & Hall/CRC

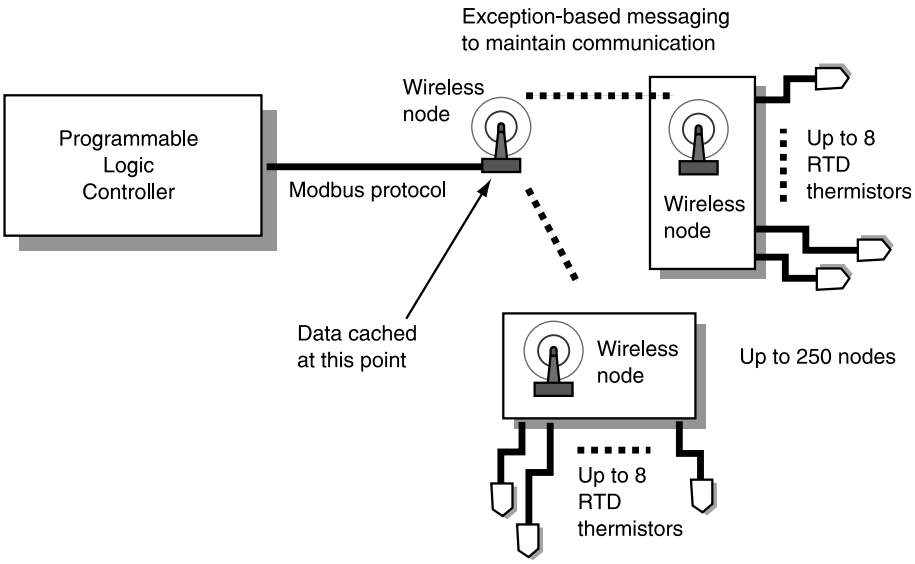


Figure 55.4. Wireless nodes and temperature sensors within heater feedback system.

constant for heat loss in this piping system was on the order of 30 min. A sampling rate that delivered either changed data readings or a notification every few minutes would have provided adequate feedback to the heating system and ensured proper operation of the pumping units. The 1 min sampling rates were a conservative approach to this application.

Figure 55.4 illustrates the organization of the wireless nodes and temperature sensors in relation to the primary controller and Modbus.

The primary controller in this example consisted of a programmable logic controller set up to respond to predefined thresholds. The simple logic progression detects when the temperature drops below a certain value and turns on the heater in the corresponding section of pipe. When the temperature rises above a specified value, the PLC turns the heater off.

55.5.2.2 The Results

The use of exception-based monitoring demonstrated in this example reduced network load and improved reporting time without re-engineering any existing system component. In a polled system, designers typically schedule queries based on a worst-case analysis, generating traffic at regular intervals regardless of whether that traffic conveys useful information. Furthermore, the detection of state changes incurs a delay that averages one-half of the polling cycle because the state change is asynchronous. In exception-based messaging, the sensor generates a message immediately, relying on the MAC strategy to determine the earliest time the message can be transmitted. Through this approach, there is no inherent delay in reporting state changes.

In the BlackGold instance, instead of generating a new message every time that it is polled, the sensor generates a message based on the relevant criteria. In this case, either the change in temperature or the periodic timeout initiates the message transfer. In the overall system, the understanding is that unless a new message has been delivered, the current temperature data that is present is considered valid. That temperature value can be considered valid up to the point at which a sensor provides a different value.

The approach used by the design team supports a higher level of scalability than the previously described example, a benefit of exception-based monitoring. The limit in this particular situation applies to the PLC device and Modbus, which together can handle a maximum of 250 end points. But, the manner in which the wireless device communicated with the system was, in effect, spoofing the Modbus, which could permit a more extensive range of sensors to be deployed than the usual address limitations. This example demonstrated an effective technique for supporting multiple parallel buses

© 2005 by Chapman & Hall/CRC

using the Modbus protocol. The implementation could have been scaled to far exceed the conventional 250-node limit on Modbus activity.

55.5.2.3 Sensor Placement

The ability of sensor components in a wireless mesh network to contain a degree of intelligence and intercommunicate can help solve potential problems that occur in monitoring situations. As an example, at one BlackGold extraction plant a technician installing temperature sensors along the pipe every 5 ft failed to notice that he had placed a sensor very close to a steam vent. Instead of an ambient temperature of −40°F, the sensor was indicating that the temperature was 40°F. This caused the heating unit for that section of pipe to turn off and eventually the fluid in the pipe gelled, causing a major shutdown of the system.

Using a mesh network design, where the nodes can communicate in a peer-based fashion, this type of problem could be eliminated through logical design. In such a design, the nodes would not only report temperature data back to the controller that turns the heater on and off, but each node would also check periodically with the neighboring nodes. Logically, in a group of nodes spaced along a 20 ft length of pipe, three of them would not be reporting −10°F while one of them is reporting 40°F. However, if the nodes are equipped with a type of voting logic, then they can identify unexpected values, and those values can be flagged and used to generate alarms to service technicians. Because the mesh provides the flexibility to communicate from measuring point to measuring point, a group of nodes can function as a buddy system where the nodes are checking up on each other in addition to their normal tasks.

55.5.3 Monitoring Cargo Shipments Using Leaf Nodes

CoolTransport ships a variety of products, including produce and pharmaceutical items that require temperature-control monitoring to prevent damage or degradation during transport. The cold chain management techniques employed by CoolTransport involve sensor placement within the cargo area to provide continuous monitoring of the ambient temperature. Through this monitoring process, the end customer can determine at the shipment destination whether the product was held at the appropriate temperature and whether or not to accept the cargo. Having used a variety of monitoring techniques, CoolTransport found an essential flaw in their approach. Monitored values from sensors implanted within the cargo could not be read without unpacking a significant amount of the shipment. If the customer decided at that point, because of sensor readings, to reject the shipment, a substantial amount of time and effort would be required to repack the cargo. For this reason, and to satisfy additional requirements, CoolTransport set out to evaluate wireless techniques for relaying the monitored sensor values to an external network. Customers could then examine these values and decide whether to accept or reject the cargo before any items had been unpacked.

The particular requirements of this application suggested a hybrid approach that incorporated elements of both mesh and star networks. Because the temperature sensors used battery power and were out of network contact during shipping, they would not function as standard mesh nodes. On the other hand, these nodes could employ point-to-point style of messaging very naturally, thus avoiding the complexity of synchronizing their sleep cycles and expending battery power to relay for other nodes. Consequently, the network design employed a standard mesh within the loading facility and *leaf node* temperature sensors with more limited participation in the network.

In this model, the temperature sensors perform no network functions during transit and merely log data. At a predetermined docking point, these leaf nodes recognize the proximity of a wireless mesh access point and automatically convey the data collected during the transport period. This data can then be used to inform the customer of the temperatures maintained during shipment and consolidated at a central point, linked through a conventional wired network, for tracking and evaluation.

As an example of the problem faced by CoolTransport, one of their contracts involved large shipments of lettuce transported during the summer when ambient air temperatures along the trucking route often reached in excess of 100°F. A shipment of lettuce represents a valuable commodity, but not

© 2005 by Chapman & Hall/CRC

an extremely valuable commodity, so the placement of one or two sensors and recording monitors within the truck's cargo area was considered sufficient to provide adequate temperature fluctuation readings to the customer. Reaching those sensors once the truck arrived at the loading dock, however, required that almost one third of the lettuce cartons be unpacked to gain access to the first sensor and its recorded data. If the customer made the decision to reject the shipment at that point, then a very large number of cartons of lettuce would have to be repacked, compounding the losses of the trucking company. A solution that could provide a full accounting of the sensor's readings during transport could save time and reduce costs for both the customer and the shipper.

55.5.3.1 Deployment Strategy and Implementation

CoolTransport embarked on an approach whereby the sensors transported with the shipment would take measurements once a minute and record the monitored values in a log. By design, when the truck pulls up to the dock where the product is being unloaded, a standard mesh network is deployed at the docking facility. A node at each of the loading bays lets the shippers pull the truck up to the loading bay, back up, and open the door, and the temperature sensor that is inside completes its 1 min wait cycle. Once that measurement time elapses, the temperature sensor identifies the network, recognizes that it is at its destination, and proceeds to register itself on the network and offload the temperature information.

The fixed network at the loading facility gets that information back through the mesh network, which transfers it to a PC or another data display station. The displayed values of the temperature record indicate whether the shipment should be accepted or rejected, based on whether the temperature remained within acceptable values during the transport period. Figure 55.5 depicts the deployment configuration used in this example.

This collected information can be transferred using broadband channels to a central location. If it is a grocery store chain, for example, then they can track it from central headquarters. From a communications standpoint, this network differs from a conventional network in that the nodes implanted in the truck operate on extremely low power. These nodes, designed to run on watch batteries, have to operate for at least a year without replacement. Each of the sensors is reused — they have to be designed for long life.

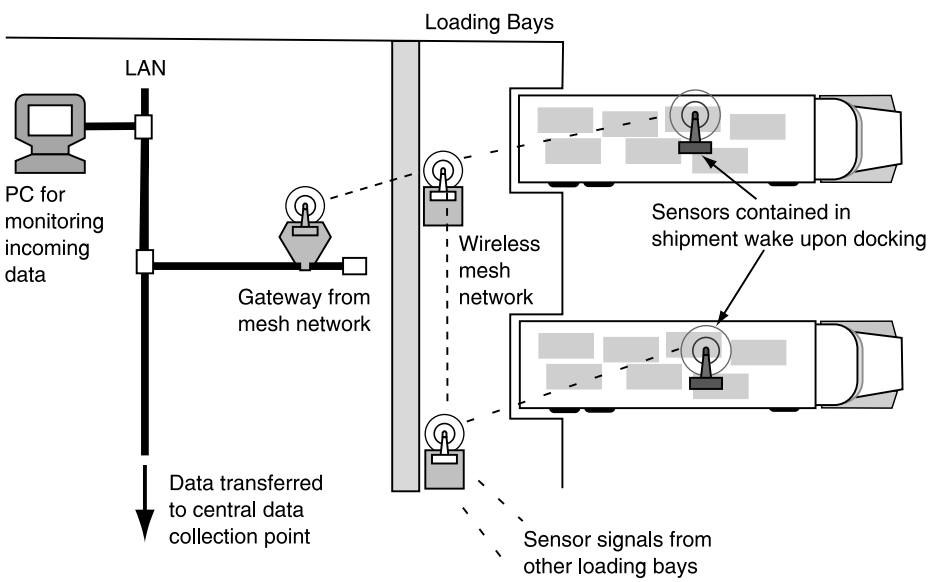


Figure 55.5. Deployment configuration for CoolTransport sensor network.

© 2005 by Chapman & Hall/CRC

The life expectancy can be maintained through a power cycle that is set up to maintain a very low duty cycle. Temperature sensors wake up once a minute, make a measurement, and listen, attempting to detect a wireless mesh network nearby. If they do not detect a network, then they go back to sleep.

The other primary difference in this approach, compared with other distributed sensor architectures, is that the data flow does not take place from trailer-based unit to trailer-based unit. Data flow is always from the trailer-based unit through the wireless mesh network to a PC or other data collection point. The nodes residing in the trucks are different from typical member nodes of the mesh network. The term *leaf nodes* has been applied to distinguish their unique characteristics.

55.5.3.2 Wireless Mesh Configurations Employing Leaf Nodes

Leaf nodes do not function as full-fledged members of the wireless mesh network. On the outside perimeter of the wireless mesh the leaf nodes can talk point-to-point, communicating to the mesh node that resides at the docking point in the bay. The node at the bay, in essence, becomes a proxy in the mesh for the nodes that are being transported in the trucks.

This technique resolves a number of issues, including the following:

- *Reduces address data overhead*. Employing leaf nodes, as handled in this example, removes the need for creating a very large address space to accommodate all of the potential addresses in the network. For low-data-rate sensor networks, dedicating a substantial amount of the transmitted data to addressing schemes is counterproductive — the overhead is an unnecessary burden. By employing dynamic address allocation, the embedded sensor can wake up to join the network and get assigned a unique identification to communicate within the network. In this example, the node at the bay serves as a proxy so that the temperature-sensing node is never exposed to the network. Communication between the node at the bay and the temperature-sensing node can be mutually agreed and the proxy communicates the temperature values associated with the sensor to the wireless mesh network. The node does not need to be assigned an ID to transfer values.
- *Allows unsynchronized sleep cycles*. Nodes that relay on behalf of their neighbors must synchronize their sleep cycles. Because of clock drift, this is a significant problem for large networks with low duty-cycles. In the case of CoolTransport's application, synchronization is particularly difficult because the temperature sensors travel between networks that may not be synchronized at all. By eliminating the relaying requirement, leaf nodes may sleep in a completely unsynchronized manner, which greatly simplifies the implementation.
- *Eliminates cargo unloading*. Monitored values can be relayed to the mesh immediately following arrival without the need to unload any of the cargo within the shipment.

The temperature sensors store data collected during transport in a low-power SRAM, which can then be reset after the cargo is unloaded in preparation for the next monitoring operation. The logic driving the monitoring operations is contained in an ASIC, which also has very low power requirements.

55.5.3.3 The Results

The deployment of the hybrid wireless mesh network using leaf nodes proved successful, providing a valuable proof of concept for this technique. The leaf node technique can be effectively applied to many different varieties of sensors, depending on the nature of the cargo and the critical sensitivities. For example, the sensor might be equipped to measure humidity, maximum *G* forces, or the presence of a particular chemical agent. The same principles can be used so that the monitored values are relayed to a proxy node at the dock upon arrival and then transferred through the wireless mesh network to a central data collection point.

This example differs from a classical sensor network structure, which usually trickles data through the network a few bytes at a time. In the CoolTransport example, the sensor stays out of communication with the network for a prolonged period, caching all data during that time. Upon docking and relinking with the wireless mesh network through the proxy, a substantial amount of data is transferred during a single hop, after which the sensor drops out of communication once again.

© 2005 by Chapman & Hall/CRC

55.5.3.4 Scenarios That Favor a Leaf Node Approach

The leaf node approach provides favorable benefits in two distinct areas:

- *Extending battery-powered applications*. In distributed sensor applications that must extend battery life for lengthy periods, a wireless mesh network presents problems in that data transmissions to neighboring nodes can consume battery reserves. This conflicts with a key strategy for extending battery life, i.e. reducing the duty cycle periods so that the sensor is running as little as possible, ideally spending long periods in sleep mode. While in sleep mode, a node cannot relay for another node. In a full-fledged mesh network, some system of coordination must be used among the nodes to control wakeup and sleep cycles. The complexity of solving this problem can be a considerable challenge in many types of wireless mesh implementation. The leaf-node approach avoids the need for time synchronization by eliminating communication with the rest of the mesh network until the time at which the cached data is transferred.
- *Minimizing address space requirements*. For a low-end distributed sensor network, even the difference between a two-byte identifier and a six-byte identifier can be a crucial difference in the utilization of bandwidth. In this example, where nodes are moving between networks, even a two-byte address is limited to some 65,000 unique addresses. However, among a number of networks that may be visited by a node with one of these unique IDs, the likelihood of encountering an identical address is unacceptably high. In this example, the proxy node circumvents the need for the leaf nodes to maintain a large address space, acting as an intermediary in the communication with the rest of the wireless mesh network.

55.5.4 Devising a Wireless Mesh Security System

This example of a wireless mesh design illustrates a more well-rounded approach to the technology, taking better advantage of the design principles discussed in Section 55.4. A security company, IronMan Security, offers an access control and security system that consists of a central logging and control station and up to 500 devices. Typical supported devices include pass-card readers, keypads, electronic door locks, and sensors. The design specifications for this system required that transactions be completed within 1 s, including accepting input from a card reader or keypad, validating the entry, and activating the corresponding lock. The control station also has 1 s to process alarm conditions, such as intrusion detection. The system must also perform continuous self-monitoring and report any device failures that occur.

The original implementation for the IronMan system relied on a proprietary protocol operating over a multi-drop RS-485 bus. To manage medium access, the network was organized using a ''master and slave'' approach. Within this network, no slave can transmit except in response to a message from the master. To satisfy the operating criteria, the control station exchanges messages with each device at least once per second. Each exchange begins with a message from the master, which consists of either a default five-byte ''status check'' message or a command, such as ''open the door.'' The default device response is a five-byte acknowledgement, but if the device has a condition to report (a user ID from a card swipe, for example, or a door-open alarm), it will send this data in a condition report. No response from the slave indicates a device failure and triggers a system failure report.

In Section 55.4.1, Table 55.1 provides the characteristics of a typical 802.15.4 wireless mesh product available today. As indicated in the following system parameters, the existing IronMan implementation requires an effective data rate of at least 282 Kbps, which significantly exceeds the 40 Kbps rate of a cost-effective mesh device.

The system parameters for the original implementation were:

- Maximum number of devices, $D = 500$
- Control station time between queries, $T_q = 500\,\mu s$
- Processing time/condition rpt (max.), $T_p = 12.5$ ms

© 2005 by Chapman & Hall/CRC

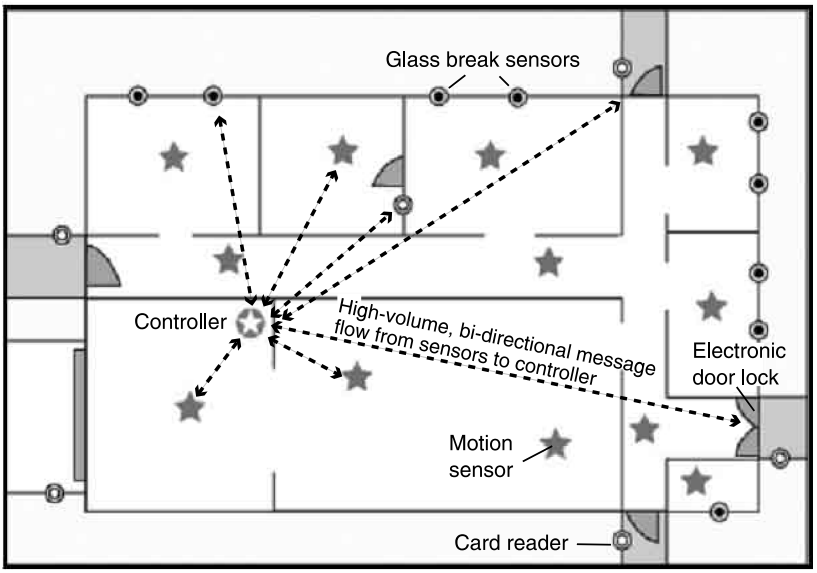


Figure 55.6. Original security system structure.

- Maximum number of condition rpts/second, $N_R = 5$
- Device response time (max.), $T_d = 1$ ms
- Total bytes exchanged/status check, $B_s = 10$

Comparing the existing IronMan system with the optimal design guidelines for wireless mesh networks, the wide disparity in the approach becomes evident. The existing design is based on a bus topology and offers no provision for the distributed control of medium access. Consequently, the system must centralize bus management in the control station and employ a query-and-response messaging model. Within this model, data cannot be pushed from the source, because the source does not know when it may transmit. Figure 55.6 shows the basic organization of the existing system.

By applying optimal mesh design principles, the handling of the condition reports can be managed more efficiently using exception-based messaging. Because each point in the wireless mesh has a built-in MAC, the query-and-response messaging to prevent bus contention can be eliminated. Rather than waiting for the next poll from the control station, a device can initiate a message as soon as it identifies a reportable condition. As a consequence, if a single condition report were the only traffic to the control station, the data rate required would be

$$\frac{RB_R}{1\,\text{s} - T_p - T_d} = 0.78\,\text{Kbps} \tag{55.1}$$

where R represents the maximum number of end-to-end retries, arbitrarily set here as $R = 3$. Because there could be as many as five such messages in the neighborhood of the control station simultaneously (which is probably a more conservative value than the actual requirement of five each second), the minimum throughput needed to support condition reports is approximately

$$0.78\,\text{Kbps} \times N_R = 3.9\,\text{Kbps} \tag{55.2}$$

Mesh devices can comfortably accommodate this data rate. Condition reports, however, are not the sole communications requirement. The self-diagnostic factor of the network is another consideration.

© 2005 by Chapman & Hall/CRC

The optimal mesh design principles state that polling from the control station is undesirable. Polling creates bottlenecks in the neighborhood of the station. To eliminate this problem, the individual devices can accomplish the necessary self-diagnostic operations.

System diagnostics can be implemented in a distributed manner by using a type of buddy system. Devices in these kinds of application naturally tend to cluster. A card reader will typically be paired with a door lock. Sensors for motion and glass breakage will usually be deployed for each room. When these devices are commissioned, they can be placed into groups of *buddies* that monitor the transmissions from each other. If a node sends a message, then the message counts as a transmission; if the node has not transmitted for a certain period of time, then it will beacon. When one of nodes in the group fails to transmit for a certain period, a neighboring node polls it. If this polling gets no response, then the neighbor can generate a condition report to alert the control station. Distributing the self-diagnostic tasks among the nodes also distributes the associated messaging. Spatial multiplexing ensures that buddy groups that are geographically separated can perform self-diagnostic operations in parallel. By this technique, the network capacity can often be replicated at each group, so individual groups can be considered independently. For such groups, the available bandwidth for self-diagnostics is approximately 90% of the total, or

$$40\,\text{Kbps} - 3.9\,\text{Kbps} = 36.1\,\text{Kbps} \tag{55.3}$$

The largest number of nodes that might compose a group is not indicated. But, assuming ten nodes in the group (and assuming that each node must transmit at least once every 0.75 s) is reasonable, this assumes a fairly aggressive 1 s reporting time, reserving 0.25 s for a possible condition report. With the beacon message occupying five bytes, the traffic load per group would be a maximum of

$$\frac{10 \times 5\,\text{bytes}}{0.75\,\text{s}} = 0.67\,\text{Kbps} \tag{55.4}$$

This value represents less than 1% of the total network capacity. This analysis becomes more difficult if the groups are in proximity, creating a requirement that they share bandwidth. Even in a worst-case deployment, however, where all 50 possible groups completely overlap, the resulting traffic will not overload the network.

55.5.5 Successful Approaches to Application Design

As shown by both the design methodologies and examples in this chapter, wireless mesh networks and distributed sensors offer a number of advantages to developers who master the techniques of working within the framework of the technology. Benefits to be gained include fault tolerance, ease of installation, incremental deployment, and greater processor efficiency. Achieving these benefits, however, requires careful attention to the architectural model used; in particular, great care should be taken when adopting the familiar centralized organization and messaging models of wired systems. Very often, these systems make tacit assumptions about the communication medium. Often, these assumptions do not apply to a practical wireless mesh application. By following the guidelines offered in this chapter, developers can construct efficient, practical applications and improve upon the design goals of wired systems.

References

[1] Poor, R. and Hodges, B., Reliable wireless networks for industrial systems, Ember Corporation Technical White Paper, http://www.ember.com/products/whitepapers.

[2] Corson, S. and Macker, J., Architectural considerations for mobile mesh networking, IETF Network Working Group, May 1996.

© 2005 by Chapman & Hall/CRC

[3] Murphy, J., Mesh networks solve distribution dilemmas, *Wireless Europe*, November 2000.
[4] Chandrakasan, A. *et al*., Design considerations for distributed microsensor systems, in *Custom Integrated Circuits Conference (CICC)*, May 1999.
[5] Poor, R., Gradient routing in *ad-hoc* networks, MIT Media Laboratory, October 2001, http://www.media.mit/pia/Research/ESP/texts/poorieeepaper.pdf.
[6] Schurgers, C. and Srivastava, M.B., Energy efficient routing in wireless sensor networks, in *MILCOM'01*, October 2001, http://www.janet.ucla.edu/~curts/papers/MILCOM01.pdf.
[7] Poor, R., Wireless embedded networking systems, SensIT PI Meeting, January 2002, http://dtsn.darpa.mil/ixo/sensit/PI_Briefs/Poor_Ember.ppt.
[8] Gupta, P. and Kumar, P.R., The capacity of wireless networks, *IEEE Transactions on Information Theory*, IT-46 (2), 388, 2000.

© 2005 by Chapman & Hall/CRC

X

Beamforming

56. Beamforming *J.C. Chen and K. Yao* *1069*
Introduction • DOA Estimation and Source Localization • Array System Performance Analysis and Robust Design • Implementations of Two Wideband Beamforming Systems

This section discusses beamforming technology. To show the importance of this technology, we have separated it from the other signal processing chapters. Beamforming uses signal processing techniques to infer information from multiple time signals. The individual time signals are collected from sensors located at different positions.

The members of Yao's group at UCLA described applications of beamforming, limitations of the approach, and how it can be used.

© 2005 by Chapman & Hall/CRC

56
Beamforming

J.C. Chen and K. Yao

56.1 Introduction

56.1.1 Historical background

Beamforming is a space–time operation in which a waveform originating from a given source but received at spatially separated sensors is coherently combined in a time-synchronous manner. If the propagation medium preserves sufficient coherency among the received waveforms, then the beamformed waveform can provide an enhanced signal-to-noise ratio (SNR) compared with a single sensor system. Beamforming can be used to determine the direction-of-arrival(s) (DOAs) and the location(s) of the source(s), as well as perform spatial filtering of two (or more) closely spaced sources. Beamforming and localization are two interlinking problems, and many algorithms have been proposed to tackle each problem individually and jointly (i.e. localization is often needed to achieve beamforming and some localization algorithms take the form of a beamformer). The earliest development of space–time processing was for enhancing SNR in communicating between the U.S. and the U.K. dating back before the World War II [1]. Phase-array antennas based upon beamforming for radar and astronomy were developed in the 1940s [2]. Since then, phase-array antennas utilizing broad ranges of radio frequencies (RFs) have been used for diverse military and civilian ground, airborne, and satellite applications. Similarly, sonar beamforming arrays have been used for more than 50 years.

Recent developments in integrated circuit technology have allowed the construction of low-cost small acoustic and seismic sensor nodes with signal processing and wireless communication capabilities that can form distributed wireless sensor network systems. These low-cost systems can be used to perform detection, source separation, localization, tracking, and identification of acoustic and seismic sources in diverse military, industrial, scientific, office, and home applications [3–7]. The design of acoustic localization algorithms mainly focuses on high performance, minimal communications load, computationally efficiency, and robust methods to reverberant and interference effects. Brandstein and Silverman [8] proposed a robust method for relative time-delay estimation by reformulating the problem as a linear regression of phase data and then estimating the time delay through minimization of a robust statistical error measure. When several signals coexist, the relative time delay of the dominant signal was shown to be effectively estimated using a second-order subspace method [9]. A recent application of particle filtering to acoustic source localization using a steered beamforming

© 2005 by Chapman & Hall/CRC

framework also promises efficient computations and robustness to reverberations [10]. Another attractive approach using the integration (or fusion) of distributed microphone arrays can yield high performance without demanding data transfer among nodes [11]. Unlike the aforementioned approaches that perform independent frame-to-frame estimation, a tracking framework has also been developed [12] to provide power-aware, low-latency location tracking that utilizes historical source information (e.g. trajectory and speed) with single-frame updates.

More recently, in cellular telephony, due to the ill-effects of multipaths and fading and the need to increase performance and data transmission rates, multiple antennas utilizing beamforming arrays have also been proposed. While several antennas at the basestations can be used, only two antennas can be utilized on hand-held mobile devices due to their physical limitation. Owing to the explosive growth of cell phones around the world, much progress is being made in both the research and technology aspects of beamforming for smart antennas.

Besides various physical phenomena, many system constraints also limit the performance of coherent array signal-processing algorithms. For instance, the system performance may suffer dramatically due to sensor location uncertainty (due to unavailable measurement in random deployment), sensor response mismatch and directivity (which may be particularly serious for some types of microphone in some geometric configurations), and loss of signal coherence across the array (i.e. widely separated microphones may not receive the same coherent signal) [13]. In a self-organized wireless sensor network, the collected signals need to be well time synchronized in order to yield good performance. These factors must be considered for practical implementation of the sensor network. In the past, most reported sensor network systems performing these processing operations usually involve custom-made hardware. However, with the advent of low-cost but quite capable processors, real-time beamforming utilizing iPAQs has been reported [14].

56.1.2 Narrowband versus Wideband Beamforming

In radar and wireless communications, the information signal is modulated on some high RF f_0 for efficient transmission purposes. In general, the bandwidth of the signal over $[0, f_s]$ is much less than the RF. Thus, the ratio of the highest to lowest transmitted frequency, $(f_0 + f_s)/(f_0 - f_s)$, is typically near unity. For example, for the 802.11b ISM wireless local-area network system, the ratio is 2.4835 GHz/2 GHz = 1.03. These waveforms are denoted as narrowband. Narrowband waveforms have a well-defined nominal wavelength, and time delays can be compensated by simple phase shifts. The conventional narrowband beamformer operating on these waveforms is merely a spatial extension of the matched filter. In the classical time-domain filtering, the time-domain signal is linearly combined with a filtering weight to achieve the desired high/low/band-pass filtering. This narrowband beamformer also combines the spatially distributed sensor collected array data linearly with the beamforming weight to achieve spatial filtering. Beamforming enhances the signal from the desired spatial direction and reduces the signal(s) from other direction(s) in addition to possible time/frequency filtering. Details on the spatial filtering aspect of this beamformer will be given in Section 56.1.3.

The movement of personnel, cars, trucks, wheeled/tracked vehicles, and vibrating machinery can all generate acoustic or seismic waveforms. The processing of seismic/vibrational sensor data is similar to that of acoustic sensors, except for the propagation medium and unknown speed of propagation. For acoustic/seismic waveforms, the ratio of the highest to lowest frequencies can be several octaves. For audio waveforms (i.e. 30 Hz–15 kHz), the ratio is about 500, and these waveforms are denoted as wideband. Dominant acoustical waveforms generated from wheeled and tracked vehicles may range from 20 Hz to 2 kHz, resulting in a ratio of about 100. Similarly, dominant seismic waveforms generated from wheeled vehicles may range from 5 to 500 Hz, also resulting in a ratio of about 100. Thus, the acoustic and seismic signals of interest are generally wideband. However, even for certain RF applications, the ratio of the highest to lowest frequencies can also be considerably greater than unity. For wideband waveforms there is no characteristic wavelength, and time delays must be obtained by

© 2005 by Chapman & Hall/CRC

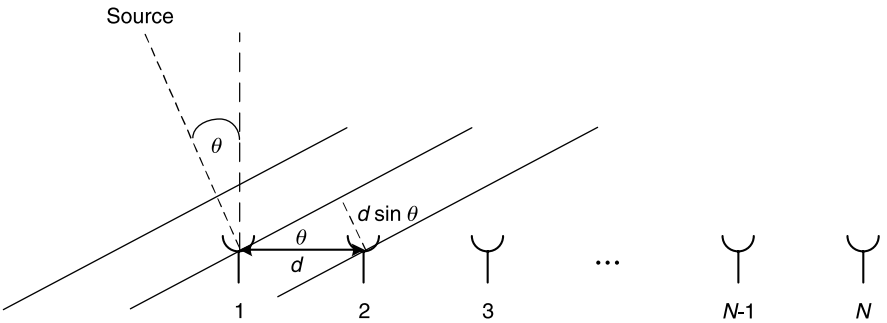


Figure 56.1. Uniform linear array of N sensors with inter-sensor spacing $d = \lambda/2$.

interpolation of the waveforms. When an acoustic or seismic source is located close to the sensors, the wavefront of the received signal is curved, and the curvature depends on the distance, then the source is in the near field. As the distances become large, all the wavefronts are planar and parallel, then the source is in the far field. For a far-field source, only the DOA angle in the coordinate system of the sensors is observable to characterize the source. A simple example is the case when the sensors are placed on a line with uniform inter-sensor spacing, as shown in Figure 56.1. Then all adjacent sensors have the same time delay and the DOA of the far-field source can be estimated readily from the time delay. For a near-field source, the collection of all relative time delays and the propagation speed of the source can be used to determine the source location. In general, wideband beamforming is considerably more complex than narrowband beamforming. Thus, the acoustic source localization and beamforming problem is challenging due to its wideband nature, near- and far-field geometry (relatively near/far distance of the source from the sensor array), and arbitrary array shape. Some basic aspects of wideband beamforming are discussed in Section 56.1.4.

56.1.3 Beamforming for Narrowband Waveforms

The advantage of beamforming for a narrowband waveform can be illustrated most simply by considering a single tone waveform

$$s(t) = a \exp(\mathrm{i}2\pi f_0 t), \quad -\infty < t < \infty \tag{56.1}$$

where $a = 1$ is the transmitted amplitude and the frequency f_0 is assumed to be fixed and known. Consider the waveforms at two receivers given by

$$x_1(t) = As(t - \tau_1) + v_1(t), \quad -\infty < t < \infty \tag{56.2}$$

$$x_2(t) = As(t - \tau_2) + v_2(t), \quad -\infty < t < \infty \tag{56.3}$$

where A denotes the received amplitude, which is assumed to be the same for both channels, τ_1 and τ_2 are the propagation times from the source to the two receivers, which are allowed to be different, and v_1 and v_2 are two received complex-valued uncorrelated zero-mean white noises of equal variance σ_0^2. Then the SNR of each receiver is given by

$$\mathrm{SNR}_{\mathrm{in}} = \mathrm{SNR}(X_1) = \mathrm{SNR}(X_2) = \frac{A^2}{\sigma_0^2} \tag{56.4}$$

© 2005 by Chapman & Hall/CRC

A beamformer is a device that combines all the received waveforms in a coherent manner. Suppose we assume the propagation delays τ_1 and τ_2 are known. In practice, these delay values may be estimated or evaluated from the geometry of the problem. Then the output of a beamformer that coherently combines the two received waveforms in Equations (56.2) and (56.3) is given by

$$\begin{aligned} y(t) &= \exp(\mathrm{i}2\pi f_0\tau_1)\, x_1(t) + \exp(\mathrm{i}2\pi f_0\tau_2)\, x_2(t) \\ &= \exp(\mathrm{i}2\pi f_0\tau_1)A\ \exp[\mathrm{i}2\pi f_0(t-\tau_1)] + \exp(\mathrm{i}2\pi f_0\tau_1)\, v_1(t) + \exp(\mathrm{i}2\pi f_0\tau_2)A\ \exp[\mathrm{i}2\pi f_0(t-\tau_2)] \\ &\quad + \exp(\mathrm{i}2\pi f_0\tau_2)\, v_2(t) \\ &= 2A\ \exp(\mathrm{i}2\pi f_0 t) + \exp(\mathrm{i}2\pi f_0\tau_1)v_1(t) + \exp(\mathrm{i}2\pi f_0\tau_2)\, v_2(t) \end{aligned} \tag{56.5}$$

The beamformer output noise variance is given by

$$E\{|\exp(\mathrm{i}2\pi f_0\tau_1)\, v_1(t) + \exp(\mathrm{i}2\pi f_0\tau_2)\, v_2(t)|^2\} = 2\sigma_0^2 \tag{56.6}$$

the beamformer output signal power is given by $(2A)^2/2$, and the beamformer output SNR is given by

$$\mathrm{SNR}_{\mathrm{output}} = \frac{(2A)^2}{2\sigma_0^2} = \frac{2A^2}{\sigma_0^2} = 2 \times \mathrm{SNR}_{\mathrm{in}} \tag{56.7}$$

Thus, Equation (56.7) shows that a coherent combining beamformer using two ideal receivers increases the effective SNR over a single receiver by a factor of 2 (i.e. 3 dB).

In the same manner, with N receivers, then Equations (56.2) and (56.3) become

$$x_n(t) = As(t-\tau_n) + v_n(t), \quad n = 1, \ldots, N, \quad -\infty < t < \infty \tag{56.8}$$

Equation (56.5) becomes

$$\begin{aligned} y(t) &= \sum_{n=1}^{N} [\exp(\mathrm{i}2\pi f_0\tau_n)x_n(t)], \quad -\infty < t < \infty \\ &= NA\ \exp(\mathrm{i}2\pi f_0 t) + \sum_{n=1}^{N} [\exp(\mathrm{i}2\pi f_0\tau_n)\, v_n(t)] \end{aligned} \tag{56.9}$$

Equation (56.6) becomes

$$E\left\{\left|\sum_{n=1}^{N}\{\exp(\mathrm{i}2\pi f_0\tau_n)\, v_n(t)\}\right|^2\right\} = N\sigma_0^2 \tag{56.10}$$

and the beamformer output SNR of Equation (56.7) now becomes

$$\mathrm{SNR}_{\mathrm{out}} = \frac{(NA)^2}{N\sigma_0^2} = N \times \mathrm{SNR}_{\mathrm{in}} \tag{56.11}$$

© 2005 by Chapman & Hall/CRC

Thus, Equation (56.11) shows in an ideal situation, when the time delays τ_n, $n = 1, \ldots, N$, are exactly known, a beamformer that performs a coherent array processing of the N received waveforms yields an SNR improvement by a factor of N relative to a single receiver.

In general, an Nth-order narrowband beamformer has the form of

$$y(t) = \sum_{n=1}^{N} w_n^* x_n(t), \quad -\infty < t < \infty \tag{56.12}$$

where $\{w_1, \ldots, w_N\}$ is a set of complex-valued weights chosen for the beamformer to meet some desired criterion. In Equation (56.9), in order to achieve coherent combining of the received narrowband waveforms, the weights are chosen so $w_n = \exp(-\mathrm{i}2\pi\tau_n)$, $n = 1, \ldots, N$.

Consider the special case of a uniform linear array, when all the N receive sensors lie on a line with an inter-sensor spacing of d, as shown in Figure 56.1. Furthermore, we assume the source is far from the linear array (i.e. in the "far-field" scenario), so the received wavefront is planar (i.e. with no curvature) and impacts the array at an angle of θ. From Figure 56.1, the wavefront at sensor 2 has to travel an additional distance of $d\sin(\theta)$ relative to the wavefront at sensor 1. The relative time delay to travel this additional distance is then given by $d\sin(\theta)/c = d\sin(\theta)/(f_0\lambda)$, where c is the speed of propagation of the wavefront and λ is the wavelength corresponding to frequency f_0. Similarly, the relative time delay of the nth sensor relative to the first sensor becomes $(n-1)d\sin(\theta)/(f_0\lambda)$. Then the time delay expression for all the sensors is given by

$$\tau_n = \tau_1 + (n-1)d\sin(\theta)/(f_0\lambda), \quad n = 1, \ldots, N \tag{56.13}$$

and the expression for all the received waveforms is given by

$$x_n(t) = A\exp(\mathrm{i}2\pi f_0 t)\exp(-\mathrm{i}2\pi f_0\tau_1)\exp(-\mathrm{i}2\pi d(n-1)\sin(\theta)/\lambda) + v_n(t), \quad n = 1, \ldots, N \tag{56.14}$$

In practice, if we use this uniform linear array, τ_1 is still known, but all the other τ_n, $n = 2, \ldots, N$, are fully known relative to τ_1 as given by Equation (56.13). Then the ideal beamformer output expression of Equation (56.9) now becomes

$$y(t) = \exp(\mathrm{i}2\pi f_0\tau_1)\left\{NA\exp(\mathrm{i}2\pi f_0 t) + \sum_{n=1}^{N}[\exp(\mathrm{i}2\pi d\sin(\theta)/\lambda)\, v_n(t)]\right\} \tag{56.15}$$

which still achieves the desired $\mathrm{SNR}_{\mathrm{out}} = N \times \mathrm{SNR}_{\mathrm{in}}$ of Equation (56.11).

Now, suppose each sensor has a uniform response in all angular directions (i.e. isotropic over $[-\pi, \pi]$). The beamformer angular transfer function for a uniform linear array with inter-sensor spacing of $d = \lambda/2$ is given by

$$\begin{aligned} H(\theta) &= \sum_{n=1}^{N}\exp[-\mathrm{i}\pi(n-1)\sin(\theta)] = \frac{1-\exp[-\mathrm{i}\pi N\sin(\theta)]}{1-\exp[-\mathrm{i}\pi\sin(\theta)]} \\ &= \frac{\exp\{-\mathrm{i}[\pi/2(N-1)\sin(\theta)]\}\,\sin[(N\pi/2)\sin(\theta)]}{\sin[(\pi/2)\sin(\theta)]}, \quad -\pi \le \theta < \pi \end{aligned} \tag{56.16}$$

© 2005 by Chapman & Hall/CRC

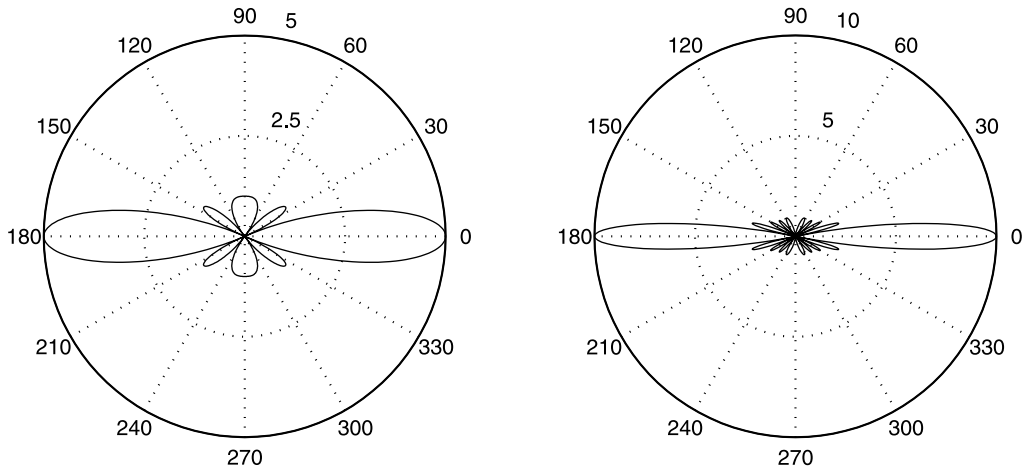


Figure 56.2. Polar plot of $|H(\theta)|$ versus θ for $d = \lambda/2$ and $N = 5$ (a) and $N = 10$ (b).

A polar plot of $|H(\theta)|$ displayed from 0 to 360° for $N = 5$ is shown in Figure 56.2(a) and for $N = 10$ is shown in Figure 56.2(b). We note, in this figure, that the linear array lies on the 90° to −90° line (in these two figures 270° = −90°) and that the gain is symmetric about this line. Thus, there is a high gain at the 0° direction (the "forward broadside" of the array) as well at the 180° direction (the "backward broadside"), with various sidelobes in other directions. In some applications we may assume that all the desired and unwanted sources are known to be in the forward sector (i.e. in the −90° to 0° to −90° sector). If the desired source is in the high gain 0° direction, then other unwanted sources in the directions of the sidelobes form interferences to the reception of the desired source. Thus, an array with a mainlobe having high gain over a narrow angular sector plus sidelobes with small values is considered desirable. From Equation (56.16) and Figure 56.2, as the number of array elements N increases, the mainlobe of the beamformer angular response becomes narrower and thus able to provide a better angular resolution, while the sidelobe peak values stay the same relative to the beam peak.

On the other hand, if we set $d = \lambda$, then Equation (56.16) has the form of

$$H(\theta) = \frac{\exp\{-\mathrm{i}[\pi(N-1)\sin(\theta)]\}\ \sin[(N\pi)\sin(\theta)]}{\sin[(\pi)\sin(\theta)]}, \quad -\pi \le \theta < \pi \tag{56.17}$$

A polar plot of this $|H(\theta)|$ again displayed from 0 to 360° for $N = 5$ is shown in Figure 56.3(a) and for $N = 10$ is shown in Figure 56.3(b). We note, in these two figures, in addition to the desired high gains in the 0° and 180°, there are also two undesired equal high gains with large angular spreads at 90° and 270°. These two additional large gains may cause a large interference to the desired source signal from unwanted sources in these directions. The spatial Nyquist criterion requires the inter-sensor spacing d of a uniform linear array to be less or equal to $\lambda/2$ to avoid grating lobes (also called spatial aliasing). This phenomenon is analogous to spectral aliasing due to the periodicity in the frequency domain created by sampling in time. Thus, a uniform linear array is most commonly operated at the $d = \lambda/2$ condition.

In Equation (56.5) we have assumed both time delays τ_1 and τ_2 are known. Then the optimum array weights are given by $w_1 = \exp(-\mathrm{i}2\pi f_0\tau_1)$ and $w_2 = \exp(-\mathrm{i}2\pi f_0\tau_2)$. For a single source and a unform linear array of N sensors, as shown in Figure 56.1, as long as the DOA angle θ of the source is known (or estimated), then the remaining relative time delays relative to the first sensor can be evaluated from the geometry. However, for two or more sources, suppose the DOA θ_1 of the desired source is known (or estimated), but other unwanted interfering source DOAs are unknown. The minimum variance desired

© 2005 by Chapman & Hall/CRC

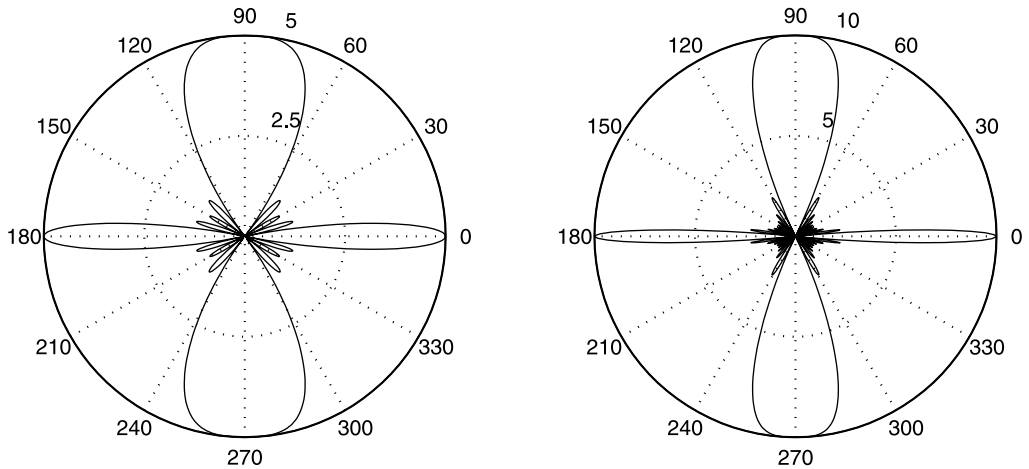


Figure 56.3. Polar plot of $|H(\theta)|$ versus θ for $d = \lambda$ and $N = 5$ (a) and $N = 10$ (b).

response (MVDR) method [15] provides a computationally attractive solution for constraining the array response of the desired source DOA angle to a fixed value (say the unity value) while minimizing the response to all other interference sources. From the output of the Nth-order beamformer in Equation (56.12), denote its $N \times N$ autocorrelation matrix as R, the array weight vector as $W = [w_1, \ldots, w_N]^{\mathrm{T}}$, and the steering vector as $C(\theta) = [1, \exp(-\mathrm{i}\theta), \ldots, \exp(-\mathrm{i}(N-1)\theta)]^{\mathrm{T}}$. Then the MVDR solution satisfies

$$\mathrm{Min}\{W^{\mathrm{H}}RW\}, \quad \text{subject to } W^{\mathrm{C}}C(\theta_1) = 1 \tag{56.18}$$

where superscript C is the complex-valued operator operator, T the transpose C operator and H is the complex conjucation operator. MVDR solution is then given by

$$W = R^{-1}C(C^{\mathrm{H}}R^{-1}C)^{-1} \tag{56.19}$$

Figure 56.4 shows the $N = 20$ uniform linear array MVDR beamformer output response (dB) versus spatial angle to a unit-amplitude single-tone source of $f = 900$ Hz at DOA of $\theta_1 = 30^\circ$, subjected to a broadband white Gaussian interferer of variance $\sigma^2 = 2$ at DOA of $\theta_2 = 60^\circ$, in the presence of additive white Gaussian noise of unity variance. The beamformer input signal-to-interference-plus-noise ratio SINR $= -3.7$ dB and the output SINR $= 10.0$ dB result in a gain of SIRN $= 13.7$ dB. In Figure 56.4, we note this MDVR beamformer achieved a unity gain at the known desired angle of $\theta_1 = 30^\circ$ as constrained and placed a null of over -40 dB at the interference angle of 60°, which is not specifically specified in the algorithm. The nulling angle information of θ_2 was obtained implicitly from the autocorrelation matrix R of the array output data.

56.1.4 Beamforming for Wideband Waveforms

Consider a source waveform containing two tones at frequency f_1 and f_2 with amplitudes a_1 and a_2 respectively, as given by

$$s(t) = a_1 \exp(\mathrm{i}2\pi f_1 t) + a_2 \exp(\mathrm{i}2\pi f_2 t), \quad -\infty < t < \infty \tag{56.20}$$

© 2005 by Chapman & Hall/CRC

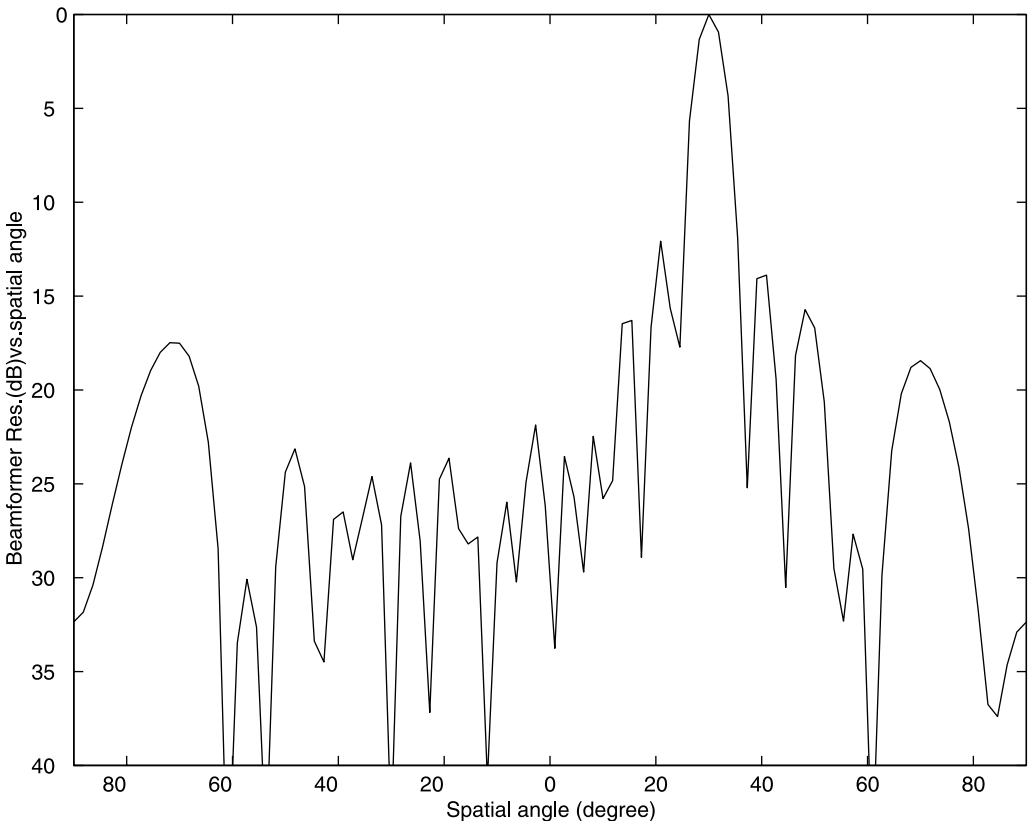


Figure 56.4. $N = 20$ uniform linear array MVDR beamformer response (dB) versus θ.

Let the wavefront of the above far-field source impact two sensors on a uniform linear array with a spacing of d. Then the received waveforms $x_1(t)$ and $x_2(t)$ have the form of

$$x_1(t) = A_1 \exp[\mathrm{i}2\pi f_1(t - \tau_1)] + A_2 \exp[\mathrm{i}2\pi f_2(t - \tau_1)] + v_1(t), \quad -\infty < t < \infty \tag{56.21}$$

$$x_2(t) = A_1 \exp[\mathrm{i}2\pi f_1(t - \tau_2)] + A_2 \exp[\mathrm{i}2\pi f_2(t - \tau_2)] + v_2(t), \quad -\infty < t < \infty \tag{56.22}$$

Suppose we want to use the narrowband beamformer as given by Equation (56.12) in the form of Figure 56.5 with $N = 2$ sensor, but only one complex weight per sensor for possible coherent combining. The issue is whether we can find two complex-valued weights $\{w_1, w_2\}$ able to achieve the desired expression on the right-hand side of Equation (56.23) given by

$$y(t) = w_1^* x_1(t) + w_2^* x_2(t) = c_1 A_1 \exp(\mathrm{i}2\pi f_1 t) + c_2 A_2 \exp(\mathrm{i}2\pi f_2 t) + v_1'(t) + v_2'(t), \quad -\infty < t < \infty \tag{56.23}$$

where c_1 and c_2 are some arbitrary complex-valued constants and $v_1'(t)$ and $v_2'(t)$ are two uncorrelated white noises. After some algebra, one can show that a narrowband beamformer with only one complex-valued weight per sensor channel cannot achieve the desired result in Equation (56.23).

Now, consider a wideband beamformer with N sensors and M complex-valued weights $\{w_{nm}, n = 1, \ldots, N, m = 1, \ldots, M\}$ shown in Figure 56.5. For the nth sensor channel, the M weights

© 2005 by Chapman & Hall/CRC

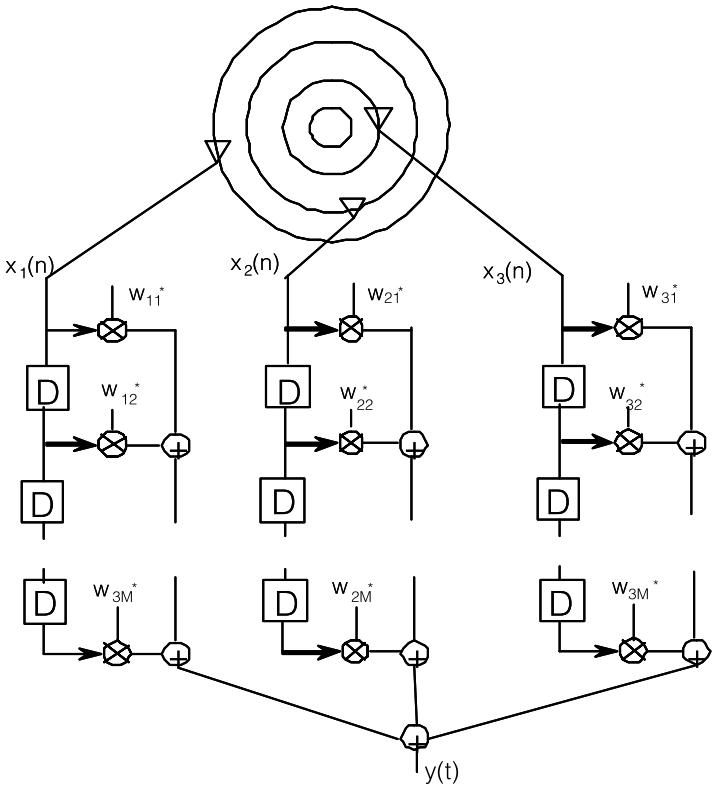


Figure 56.5. Wideband beamformer with $N = 3$ sensors and M taps per sensor section.

$\{w_{nm}, m = 1, \ldots, M\}$ with the $(M - 1)$ time delays of value T form an Mth-order tapped delay line. For the time sampled case, they form an Mth-order finite impulse response (FIR) filter with the time delay T replaced by Z^{-1}. Thus, a wideband beamformer performs spatial–time–frequency filtering operations. In the above case (considered in Equation (56.23)), it can be shown (after some algebra) that a wideband beamformer with $N = 2$ and $M = 2$ (i.e. two complex-valued weights per sensor channel) can achieve the desired coherent combining. In general, it can be shown that, for N tones with distinct frequencies, a wideband beamformer using the uniform linear array of N sensors needs N complex-valued weights per sensor channel to achieve the desired coherent combining. Of course, in practice, a realistic wideband waveform is equivalent to an infinite number of tones in the $[f_{\text{low}}, f_{\text{high}}]$ band. Thus, a sufficiently large M number of complex-valued weights per sensor channel in the wideband beamformer may need to be used to approximate the desired coherent combining effect. The number of sensors N is mainly used to control the narrowness of the mainlobe, similar to that in the narrowband beamformer case seen in the Section 56.1.3.

In Section 56.2 we deal with DOA estimation and source localization. Then, in Section 56.3 we show the array system performance analysis and robust design. Finally, in Section 56.4 we demonstrate practical implementation of two wideband beamforming systems.

56.2 DOA Estimation and Source Localization

DOA, bearing estimation, or angle of arrival (AOA), and source localization are the classical problems in array signal processing. These problems are essential in many RF and acoustic/seismic applications, and a variety of algorithms have been developed over the years. In most RF and sonar applications, only the DOA estimation is of interest due to the far-field geometry between the platform and the target,

© 2005 by Chapman & Hall/CRC

and the term ''source localization'' has been used interchangeably with DOA estimation. In other applications, such as acoustic and seismic sensing in a sensor network, the target may be in the near field of the sensor array, and in this case the range and the angle of the target are included as the estimation parameters. In this section we refer to near-field angle and range estimation as source localization, and far-field-angle-only estimation as DOA estimation.

The DOA estimation and source localization belong to the general class of parameter estimation problems, and in most cases closed-form solutions are not available. The algorithms can be generally categorized into two classes: parametric and time-delay methods. The parametric methods normally are based on optimizing the parameters directly from the signal model. Some examples of this type include maximum-likelihood (ML) and super-resolution subspace methods such as MUSIC [16]. Iterative algorithms are usually used to obtain the solution efficiently, or in other cases a grid-search solution can be obtained when only a crude estimation is needed. An alternative approach, which we refer to as the time-delay method, is also often used in many acoustic/seismic applications. In this case, the problem is broken down into two estimation steps. First, the relative time delays among the sensors are estimated based on some form of correlation [17]. Then, based on the estimated relative time delays, the DOA or the location of the source is estimated using some form of least-squares (LS) fit. This approach, compared with the parametric approach, has a major computational advantage, since the LS estimation can be obtained in closed form. Even though the relative time-delay estimation step is not closed form, its solution is much easier to obtain compared with the direct parametric solution. In terms of performance, the time-delay methods are suboptimal compared with the optimum parametric ML method, but in some cases they are satisfactory. However, in the case of multiple sources of comparable strength emitting simultaneously, the parametric algorithms can be expanded to estimate the DOAs or locations of the sources jointly, while the relative time-delay estimation for multiple sources is not at its mature stage [16].

The wideband extension of the ML solution has been shown to be effective [19]; however, many other suboptimal methods, such as MUSIC or its variants, do not show promising results when extended to the wideband problem. In the following, we formulate some of the DOA and source localization algorithms of the RF and acoustic/seismic problems.

56.2.1 RF Signals

For RF applications, the parametric methods are mostly used. For narrowband RF signals, the relative time delay is merely the phase difference between sensors. This narrowband property naturally lends itself to the parametric methods, and a variety of efficient algorithms have been developed. Krim and Viberg [20] have provided an excellent review and comparison of many classical and advanced parametric narrowband techniques up to 1996. Early work in DOA estimation includes the early version of the ML solution, but it did not become popular owing to its high computational cost. Concurrently, a variety of suboptimal techniques with reduced computations have dominated the field. The more well-known techniques include the minimum variance, the MUSIC, and the minimum norm. The MUSIC algorithm is perhaps one of the most popular suboptimal techniques. It provides super-resolution DOA estimation in a spatial pseudo-spectral plot by utilizing the orthogonality between the signal and noise subspaces. However, a well-known problem with some of these suboptimal techniques occurs when two or more sources are highly correlated. This may be caused by multipaths or intentional jamming, and most of the suboptimal techniques have difficulties without reverting to advanced processing or constraints. Many variants of the MUSIC algorithm have been proposed to combat signal correlation and to improve performance.

When M spatially separated RF signals impinge on an array, one snapshot of the signal received by the array can be characterized by

$$\mathbf{x}(t) = \sum_{m=1}^{M} s_m(t)\,\mathbf{d}_m(\boldsymbol{\Theta}) + \mathbf{n}(t) \tag{56.24}$$

© 2005 by Chapman & Hall/CRC

where $s_m(t)$ is the mth RF signal received by the reference sensor, $\mathbf{d}_m(\boldsymbol{\Theta})$ is the steering vector that contains the relative phase of the mth signal traversing across the array, $\boldsymbol{\Theta}$ is the parameter vector of interest, which in this case contains the DOAs of the sources, and $\mathbf{n}(t)$ is the noise vector of the array. For statistical estimation of the DOA, multiple independent snapshots of the array signal are collected. Denote L as the total number of snapshots. By the classical estimation theory, the estimation performance improves as L increases; however, in some systems the source could be maneuvering in time, thus L should be chosen to be just large enough before the DOA changes.

Depending on the system characteristics and problem at hand, the stochastic properties of the array signal determine which estimation method should be used. For most systems the noise can be characterized by a vector Gaussian process, and the covariance matrix of the signal $\mathbf{R_n} = E[\mathbf{nn}^{\mathrm{H}}]$, usually also assumed to be white $\mathbf{R_n} = \sigma^2\mathbf{I}$, can be estimated by its time averaged version $\widetilde{\mathbf{R}}_\mathbf{n}$. The correlation matrix of the data is then given by

$$\mathbf{R_x} = E\left[\mathbf{xx}^{\mathrm{H}}\right] = \mathbf{D}(\boldsymbol{\Theta})\mathbf{S}(t)\mathbf{D}(\boldsymbol{\Theta})^{\mathrm{H}} + \mathbf{R_n} \tag{56.25}$$

where $\mathbf{D}(\boldsymbol{\Theta}) = [\mathbf{d}_1(\boldsymbol{\Theta}) \cdots \mathbf{d}_M(\boldsymbol{\Theta})]$ is the steering matrix, $\mathbf{S}(t) = E[\mathbf{ss}^{\mathrm{H}}]$ is the source correlation matrix, and $s = [s_1(t) \cdots s_M(t)]^{\mathrm{T}}$ is the source vector. When the number of sources is less than the number of sensors, the data correlation matrix can be broken down into two orthogonal subspaces:

$$\mathbf{R_x} = E\left[\mathbf{xx}^{\mathrm{H}}\right] = \mathbf{U}_\mathrm{s}\boldsymbol{\Lambda}_\mathrm{s}\mathbf{U}_\mathrm{s}^{\mathrm{H}} + \mathbf{U}_\mathrm{n}\boldsymbol{\Lambda}_\mathrm{n}\mathbf{U}_\mathrm{n}^{\mathrm{H}} \tag{56.26}$$

where $\mathbf{U}_\mathrm{s}$ and $\boldsymbol{\Lambda}_\mathrm{s}$ are the matrices containing the signal subspace eigenvectors and eigenvalues, respectively, and $\mathbf{U}_\mathrm{n}$ and $\boldsymbol{\Lambda}_\mathrm{n}$ are those of the noise subspace. Define the projections into the signal and noise subspaces as

$$\boldsymbol{\Pi} = \mathbf{U}_\mathrm{s}\mathbf{U}_\mathrm{s}^{\mathrm{H}} = \mathbf{D}(\mathbf{D}^{\mathrm{H}}\mathbf{D})^{-1}\mathbf{D}^{\mathrm{H}} \tag{56.27}$$

$$\boldsymbol{\Pi}^{\perp} = \mathbf{U}_\mathrm{n}\mathbf{U}_\mathrm{n}^{\mathrm{H}} = \mathbf{I} - \mathbf{D}(\mathbf{D}^{\mathrm{H}}\mathbf{D})^{-1}\mathbf{D}^{\mathrm{H}} \tag{56.28}$$

respectively. The subspace estimation methods make use of the orthogonality between the signal and noise subspaces. In particular, the MUSIC spatial spectrum can be produced by generating the following plot:

$$P_{\mathrm{MUSIC}}(\theta) = \frac{\mathbf{a}(\theta)^{\mathrm{H}}\mathbf{a}(\theta)}{\mathbf{a}(\theta)^{\mathrm{H}}\widehat{\boldsymbol{\Pi}}^{\perp}\mathbf{a}(\theta)} \tag{56.29}$$

where $\mathbf{a}(\theta)$ is the steering vector of the array pointing to a particular direction θ and $\widehat{\boldsymbol{\Pi}}^{\perp}$ is the estimated version of the noise subspace projection matrix. Owing to the orthogonality of the array steering vector and the noise subspace, the MUSIC spectrum should generate peaks in the vicinity of the true DOAs and the algorithm then becomes a peak searching one. The MUSIC algorithm is a powerful estimator, and it is computationally attractive since it reduces the search space into a one-dimensional one. However, it usually requires almost perfect environment and conditions, and it may not be able to yield good results in practical situations. A more optimum solution, despite possibly being computationally extensive, is the ML approach. The deterministic ML (DML) solution can be given when the statistics of the source are assumed to be unknown and deterministic. This involves minimizing the following metric:

$$\widehat{\boldsymbol{\Theta}}^{\mathrm{DML}} = \arg\min_{\boldsymbol{\Theta}} \mathrm{Tr}\left(\boldsymbol{\Pi}_\mathbf{A}^{\perp}\widehat{\mathbf{R}}_\mathbf{x}\right) \tag{56.30}$$

© 2005 by Chapman & Hall/CRC

where Tr(·) is the trace operation, and where

$$\mathbf{A} = [\mathbf{a}_1(\theta_1) \cdots \mathbf{a}_M(\theta_M)] \tag{56.31}$$

$$\mathbf{\Pi}_{\mathbf{A}} = \mathbf{A}\left(\mathbf{A}^{\mathrm{H}}\mathbf{A}\right)^{-1}\mathbf{A}^{\mathrm{H}} \tag{56.32}$$

$$\mathbf{\Pi}_{\mathbf{A}}^{\perp} = \mathbf{I} - \mathbf{\Pi}_{\mathbf{A}} \tag{56.33}$$

and $\widehat{\mathbf{R}}_{\mathbf{x}}$ is the sample data correlation matrix obtained by time averaging. In certain cases the statistics of the source are known or can be estimated, and the statistical ML (SML) solution can be obtained instead [21]. The performance of the SML normally outperforms that of the DML.

56.2.2 Acoustic/Seismic Signals

56.2.2.1 Parametric Methods

The parametric methods developed for narrowband signals have also been extended to wideband signals such as acoustic and seismic. For wideband signals, the time delay becomes a linear phase in frequency. Thus, the narrowband signal model can be applied to each frequency snapshot of the wideband signal spectrum. Then, a composite wideband signal model that includes all relevant frequency components of this signal can be used instead. For an array of R microphones (or seismic sensors) simultaneously receiving M independent, spatially separated sound (or seismic) signals ($M < R$), the acoustic (or seismic) waveform arriving at the rth microphone is given by

$$x_r(t) = \sum_{m=1}^{M} h_r^{(m)}(t) \star s_m(t) + n_r(t) \tag{56.34}$$

for $r = 1, \ldots, R$, where s_m is the mth source signal, $h_r^{(m)}$ is the impulse response from the mth source to the rth sensor (i.e. a delta function in free space corresponding to the time delay or a filtered response to include the reverberation effect), n_r is the additive noise, and $\star$ denotes the convolution operation. For each chosen frame time (a function of the source motion and signal bandwidth), the received signal is appropriately digitized and collected into a space–time data vector $\mathbf{x} = [x_1(0), \ldots, x_1(L-1), \ldots, x_R(0), \ldots, x_R(L-1)]^{\mathrm{T}}$ of length RL. The corresponding frequency spectrum data vector is then given by $\mathbf{X}(\omega_k) = [X_1(\omega_k), \ldots, X_R(\omega_k)]^{\mathrm{T}}$, for $k = 0, \ldots, N-1$, where N is the number of fast Fourier transform (FFT) bins.

Denote $\mathbf{\Theta}$ as the estimation parameter: in the near-field case this is the source location vector $[\mathbf{r}_{s_1}^{\mathrm{T}}, \ldots, \mathbf{r}_{s_M}^{\mathrm{T}}]^{\mathrm{T}}$ and $\mathbf{r}_{s_m}$ is the mth source location; in the far-field case this is the angle vector $\left[\phi_s^{(1)}, \theta_s^{(1)}, \ldots, \phi_s^{(M)}, \theta_s^{(M)}\right]^{\mathrm{T}}$, and $\phi_s^{(m)}$ and $\theta_s^{(m)}$ are the azimuth and elevation angles of the mth source respectively. In general, the ML estimation of parameter $\mathbf{\Theta}$ with additive white Gaussian noise assuming idealistic nonreverberant environment is given by

$$\widehat{\mathbf{\Theta}}^{\mathrm{ML}} = \arg\max_{\mathbf{\Theta}} J(\mathbf{\Theta}) = \arg\max_{\mathbf{\Theta}} \sum_{k=1}^{N/2} W(k) \|\mathbf{P}(k, \mathbf{\Theta})\mathbf{X}(\omega_k)\|^2 \tag{56.35}$$

where [19]: $W(k)$ is the weighting function by design (e.g. lower for significant bins and stronger weighting on dominant bins and/or high frequency bins); $\mathbf{P}(k, \mathbf{\Theta}) = \mathbf{D}(k, \mathbf{\Theta})\mathbf{D}^{\dagger}(k, \mathbf{\Theta})$ is the projection matrix that projects the data into the parameter space; $\mathbf{D}^{\dagger}(k, \mathbf{\Theta}) = (\mathbf{D}(k, \mathbf{\Theta})^{\mathrm{H}}\ \mathbf{D}(k, \mathbf{\Theta}))^{-1}\mathbf{D}(k, \mathbf{\Theta})^{\mathrm{H}}$ is the pseudo-inverse of the steering matrix; $\mathbf{D}(k, \mathbf{\Theta}) = [\mathbf{d}^{(1)}(k, \mathbf{\Theta}), \ldots, \mathbf{d}^{(M)}(k, \mathbf{\Theta})]$ is the steering matrix;

© 2005 by Chapman & Hall/CRC

$\mathbf{d}^{(m)}(k,\boldsymbol{\Theta}) = [d_1^{(m)}(k,\boldsymbol{\Theta}), \ldots, d_R^{(m)}(k,\boldsymbol{\Theta})]^{\mathrm{T}}$ is the mth steering vector; $d_r^{(m)}(k,\boldsymbol{\Theta}) = \mathrm{e}^{(-\mathrm{j}2\pi k t_r^{(m)}/N)}$ is one component of the mth steering vector; $t_r^{(m)}$ is the time delay for the mth signal to traverse to the rth sensor; and only the positive frequency bins are considered (negative frequencies are simply mirror images for real-valued signals).

Note, a closed-form solution is not available for Equation (56.35) and a brute-force search is prohibitive as M increases. Efficient iterative computational methods including the alternating projection [4,19], particle filtering [10], and the SAGE method [22] have been shown to be effective both in computer simulations and in real-life data for up to $M = 2$. The nice feature about the objective function in Equation (56.35) is that it is continuous and differentiable; thus, many efficient gradient methods can be applied and tailored to the specific application for fast convergence. However, the gradient approach does not guarantee convergence to the global maximum, as the objective function is multi-modal (with multiple local extrema) in general, and the selection of the initial condition may greatly impact the estimation performance. The dependency of the initial condition is de-emphasized for two sources using the alternating projection approach [19]; but, without prior information, the above algorithms may get trapped in local solutions that may be far away from the actual location(s). For robust solutions that can achieve the global maximum, a genetic algorithm and simulated annealing can be used at the cost of convergence speed.

In most practical cases of ground sensor networks, the number of sources of interest M may not be a large number since only the nearby sources will have high enough energy to contribute to the overall received power. The interfering sources that are far away and weak will only contribute to the noise floor of the system, and thus will not be of interest to the estimator. However, in the case where the number of sources M becomes large, the computational time of the estimator may be burdensome even for the aforementioned iterative methods. For M sources of roughly equal strength, the search dimension can be limited to only that of a single source, and the highest M humps can be used as the suboptimal solutions. Obviously, this approach has several drawbacks, despite its significant computational advantage as M increases. It requires the M sources to be widely separated so the interaction among them is minimal. With some difference in strength, it is also difficult to distinguish a sidelobe of a stronger source from the presence of a weaker source. The widely used narrowband super-resolution methods such as MUSIC have been extended to the wideband case to achieve high resolution with low interaction among sources. Many have proposed using focusing matrices to transform the wideband signal subspaces into a predefined narrowband subspace [23]. Then, the MUSIC algorithm can be applied afterwards as if it was the narrowband case. This class of methods is referred to as the coherent signal subspace method (CSM). However, the drawback of these methods is the preprocessing that must be performed. For instance, the CSM methods need to construct the focusing matrices that require focusing angles that are not far from the true DOAs. A simpler noncoherent wideband MUSIC algorithm without such a condition has also been compared [19] and shown to yield poor estimation results (especially in range for the near-field case) for a finite number of data samples (limited due to moving source) and low SNR.

Alternating Projection To estimate the source location or DOA of multiple sources, the alternating projection is shown to be the most effective way. The alternating projection approach breaks the multi-dimensional parameter search into a sequence of single-source parameter searches, and yields a fast convergence rate. The following describes the alternating projection algorithm for the case of two sources, and it can be easily extended to the case of M sources. Let $\boldsymbol{\Theta} = [\boldsymbol{\Theta}_1^{\mathrm{T}}, \boldsymbol{\Theta}_2^{\mathrm{T}}]^{\mathrm{T}}$ be either the source locations in the near-field case or the DOAs in the far-field case. The *alternating projection algorithm* is as follows:

Step 1. Estimate the location/DOA of the stronger source on a single source grid

$$\boldsymbol{\Theta}_1^{(0)} = \arg\max_{\boldsymbol{\Theta}_1} J(\boldsymbol{\Theta}_1) \tag{56.36}$$

© 2005 by Chapman & Hall/CRC

Step 2. Estimate the location/DOA of the weaker source on a single source grid under the assumption of a two-source model while keeping the first source location estimate from step 1 constant

$$\boldsymbol{\Theta}_2^{(0)} = \arg\max_{\boldsymbol{\Theta}_2} J\left(\left[\boldsymbol{\Theta}_1^{(0)^{\mathrm{T}}}, \boldsymbol{\Theta}_2^{\mathrm{T}}\right]^{\mathrm{T}}\right) \tag{56.37}$$

For $i = 1, \ldots$ (repeat steps 3 and 4 until convergence).

Step 3. Iterative approximated ML (AML) parameter search (direct search or gradient) for the location/DOA of the first source while keeping the estimate of the second source location from the previous iteration constant

$$\boldsymbol{\Theta}_1^{(i)} = \arg\max_{\boldsymbol{\Theta}_1} J\left(\left[\boldsymbol{\Theta}_1^{\mathrm{T}}, \boldsymbol{\Theta}_2^{(i-1)^{\mathrm{T}}}\right]^{\mathrm{T}}\right) \tag{56.38}$$

Step 4. Iterative AML parameter search (direct search or gradient) for the location/DOA of the second source while keeping the estimate of the first source location from step 3 constant

$$\boldsymbol{\Theta}_2^{(i)} = \arg\max_{\boldsymbol{\Theta}_2} J\left(\left[\boldsymbol{\Theta}_1^{(i)^{\mathrm{T}}}, \boldsymbol{\Theta}_2^{\mathrm{T}}\right]^{\mathrm{T}}\right) \tag{56.39}$$

56.2.2.2 ML Beamforming

The main purpose of beamforming is to improve the SINR, which is often performed after a desired source location is obtained (except for the blind beamforming methods). In the most general sense of digital wideband beamforming in the time domain, the digitized received array signal is combined with appropriate delays and weighting to form the beamformer output

$$y(n) = \sum_{r=1}^{R} \sum_{\ell=0}^{L-1} w_{r\ell} x_r(n - \ell) \tag{56.40}$$

where $w_{r\ell}$ is the chosen beamforming weight to satisfy some criterion, and x_r here denotes the digitized version of the received signal. Numerous criteria exist in the design of the beamforming weight, including maximum SINR with frequency and spatial constraints. Other robust blind beamforming methods have also been proposed to enhance SINR without the knowledge of the sensor responses and locations. For instance, the blind maximum power (MP) beamformer [9] obtains array weights from the dominant eigenvector (or singular vector) associated with the largest eigenvalue (or singular value) of the space–time sample correlation (or data) matrix. This approach not only collects the maximum power of the dominant source, but also provides some rejection of other interferences and noise.

In some cases, especially for multiple sources, frequency-domain beamforming may be more attractive for acoustic signals due to their wideband nature. This is especially advantageous when an ML localization algorithm is used *a priori*, since the beamforming output is a direct result of the ML source signal vector estimate $\widehat{\mathbf{S}}^{\mathrm{ML}}(\omega_k)$ given by

$$\mathbf{Y}(\omega_k) = \widehat{\mathbf{S}}^{\mathrm{ML}}(\omega_k) = \mathbf{D}^{\dagger}(k, \widehat{\boldsymbol{\Theta}}^{\mathrm{ML}})\mathbf{X}(\omega_k) \tag{56.41}$$

© 2005 by Chapman & Hall/CRC

where $\mathbf{Y}(\omega_k)$ is the beamformed spectrum vector for the M sources [19]. The ML beamformer in effect performs signal separation by utilizing the physical separation of the sources, and for each source signal the SINR is maximized in the ML sense. When only a single source exists, $\mathbf{D}^{\dagger}(k, \widehat{\boldsymbol{\Theta}}^{\mathrm{ML}})$ degenerates to a vector and only the SNR is maximized.

56.2.2.3 Time-Delay-Type Methods

The problem of determining the location of a source given a set of differential time delays between sensors has been studied for many years. These techniques include two independent steps, namely estimating relative time delays between sensor data and then source location based on the relative time-delay estimates. Closed-form solutions can be derived for the second step based on spherical interpolation [24,25], hyperbolic intersection [26], or linear intersection [27]. However, these algorithms require the knowledge of the speed of propagation (e.g. the acoustic case). Yao and co-workers [9,28] derived closed-form LS and constrained LS (CLS) solutions for the case of unknown speed of propagation (e.g. the seismic case). The CLS method improves the performance from that of the LS method by forcing an equality constraint on two components of the unknowns. When the speed of propagation is known, the LS method of Yao and associates [9,28], after removing such unknowns, becomes the method independently derived later by Huang *et al.* [29]. Huang *et al.* [29] show that their method is mathematically equivalent to the spherical interpolation technique [24,25], but with less computational complexity. We refer to the method of Huang *et al.* [29] as the LS method of known speed of propagation and use it to compare with the other methods in this section. A similar LS formulation is also available for DOA estimation [30]. In theory, these two-step methods can work for multiple sources as long as the relative time delays can be estimated from the data. However, this remains to be a challenging task in practice. Recently, the use of higher order statistics to estimate the relative time delays for multiple independent sources has been studied [8], but its practicality is yet to be shown.

Denote the source location in Cartesian by $\mathbf{r}_s = [x_s, y_s, z_s]^{\mathrm{T}}$ and the rth sensor location by $\mathbf{r}_r = [x_r, y_r, z_r]^{\mathrm{T}}$. Without loss of generality, we choose $r = 1$ as the reference sensor for differential time delays. Let the reference sensor be the origin of the coordinate system for simplicity. The speed of propagation v in this formulation can also be estimated from the data. In some problems, v may be considered to be partially known (e.g. acoustic applications), while in others it may be considered to be unknown (e.g. seismic applications). The differential time delays for R sensors satisfy

$$t_{r1} = t_r - t_1 = \frac{\|\mathbf{r}_s - \mathbf{r}_r\| - \|\mathbf{r}_s - \mathbf{r}_1\|}{v} \tag{56.42}$$

for $r = 2, \ldots, R$. This is a set of $R-1$ nonlinear equations, which makes finding its solution $\mathbf{r}_s$ nontrivial.

To simplify the estimation to a linear problem, we make use of the following:

$$\|\mathbf{r}_s - \mathbf{r}_r\|^2 - \|\mathbf{r}_s\|^2 = \|\mathbf{r}_r\|^2 - 2\left(x_s x_r + y_s y_r + z_s z_r\right) \tag{56.43}$$

The left-hand side of Equation (56.43) is equivalent to

$$(\|\mathbf{r}_s - \mathbf{r}_r\| - \|\mathbf{r}_s\|)(\|\mathbf{r}_s - \mathbf{r}_r\| + \|\mathbf{r}_s\|) = v t_{r1}(2\|\mathbf{r}_s\| + v t_{r1}) \tag{56.44}$$

in the case of $\mathbf{r}_1 = \mathbf{0}$. Upon combining both expressions, we have the following linear relation for the rth sensor:

$$\mathbf{r}_r^{\mathrm{T}}\mathbf{r}_s + v t_{r1}\|\mathbf{r}_s\| + \frac{v^2 t_{r1}^2}{2} = \frac{\|\mathbf{r}_r\|^2}{2} \tag{56.45}$$

© 2005 by Chapman & Hall/CRC

With R sensors, we formulate the LS solution by putting $R-1$ linear equations into the following matrix form:

$$\mathbf{A}\mathbf{y} = \mathbf{b} \tag{56.46}$$

where

$$\mathbf{A} = \begin{bmatrix} \mathbf{r}_2^{\mathrm{T}} & t_{21} & t_{21}^2/2 \\ \mathbf{r}_3^{\mathrm{T}} & t_{31} & t_{31}^2/2 \\ \vdots & \vdots & \vdots \\ \mathbf{r}_R^{\mathrm{T}} & t_{R1} & t_{R1}^2/2 \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} \mathbf{r}_s \\ v\|\mathbf{r}_s\| \\ v^2 \end{bmatrix}, \quad \mathbf{b} = \frac{1}{2}\begin{bmatrix} \|\mathbf{r}_2\|^2 \\ \|\mathbf{r}_3\|^2 \\ \vdots \\ \|\mathbf{r}_R\|^2 \end{bmatrix} \tag{56.47}$$

For 3-D uncertainty, the dimension of $\mathbf{A}$ is $(R-1) \times 5$. In the case of six or more sensors, the pseudo-inverse of matrix $\mathbf{A}$ is given by

$$\mathbf{A}^{\dagger} = \left(\mathbf{A}^{\mathrm{T}}\mathbf{A}\right)^{-1}\mathbf{A}^{\mathrm{T}} \tag{56.48}$$

The LS solution for the unknown vector can be given by $\mathbf{y} = \mathbf{A}^{\dagger}\mathbf{b}$. The source location estimate is given by the first three elements of $\mathbf{y}$ and the speed of propagation estimate is given by the square-root of the last element of $\mathbf{y}$.

In the three-dimensional case there are five unknowns in $\mathbf{w}$. To obtain an overdetermined solution, we need at least five independent equations, which can be derived from the data of six sensors. However, placing sensors randomly does not provide much assurance against ill-conditioned solutions. The preferred approach would be to use seven or more sensors, yielding six or more relative delays, and to perform an LS fitting of the data. In the two-dimensions problem, the minimum number of sensors can be reduced by one. If the propagation speed is known, then the minimum number of sensors can be further reduced by one.

Notice in the unknown vector $\mathbf{y}$ of Equation (56.47) that the speed of propagation estimate can also be given by

$$\widehat{v} = \frac{\widehat{v\|\mathbf{r}_s\|}}{\|\widehat{\mathbf{r}}_s\|} \tag{56.49}$$

using the fourth and the first three elements of $\mathbf{y}$. To exploit this relationship, we can add another nonlinear constraint to ensure equivalence between the speed of propagation estimates from the fourth and the fifth elements. By moving the fifth element of $\mathbf{y}$ to the other side of the equation, we can rewrite Equation (56.46) as follows:

$$\mathbf{A}\mathbf{y} = \mathbf{b} + v^2\mathbf{d} \tag{56.50}$$

where

$$\mathbf{A} = \begin{bmatrix} \mathbf{r}_2^{\mathrm{T}} & t_{21} \\ \mathbf{r}_3^{\mathrm{T}} & t_{31} \\ \vdots & \vdots \\ \mathbf{r}_R^{\mathrm{T}} & t_{R1} \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} \mathbf{r}_s \\ v\|\mathbf{r}_s\| \end{bmatrix} \tag{56.51}$$

© 2005 by Chapman & Hall/CRC

$$\mathbf{b} = \frac{1}{2}\begin{bmatrix} \|\mathbf{r}_2\|^2 \\ \|\mathbf{r_3}\|^2 \\ \vdots \\ \|\mathbf{r}_R\|^2 \end{bmatrix}, \quad \mathbf{d} = -\frac{1}{2}\begin{bmatrix} t_{21}^2 \\ t_{31}^2 \\ \vdots \\ t_{R1}^2 \end{bmatrix} \tag{56.52}$$

In this case, the dimension of $\mathbf{A}$ is $(R-1) \times 4$ for three-dimensional uncertainty. The pseudo-inverse of matrix $\mathbf{A}$ is given by

$$\mathbf{A}^\dagger = \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T. \tag{56.53}$$

The CLS solution for the unknown vector can be given by $\mathbf{y} = \mathbf{A}^\dagger\mathbf{b} + v^2\mathbf{A}^\dagger\mathbf{d}$. Define $\mathbf{p} = \mathbf{A}^\dagger\mathbf{b}$ and $\mathbf{q} = \mathbf{A}^\dagger\mathbf{d}$. The source location and speed of propagation estimates can be given by

$$\begin{aligned} x_s &= p_1 + v^2 q_1 \\ y_s &= p_2 + v^2 q_2 \\ z_s &= p_3 + v^2 q_3 \\ v\|\mathbf{r_s}\| &= p_4 + v^2 q_4 \end{aligned} \tag{56.54}$$

where p_i and q_i are the ith entries of $\mathbf{p}$ and $\mathbf{q}$ respectively. The number of unknowns appears to be five, but the five unknowns only contribute four degrees of freedom due to the following nonlinear relationship:

$$\|\mathbf{r}_s\|^2 = x_s^2 + y_s^2 + z_s^2 \tag{56.55}$$

By substituting (56.54) into (56.55), the following third order constraint equation results

$$\alpha(v^2)^3 + \beta(v^2)^2 + \gamma(v^2) + \delta = 0 \tag{56.56}$$

where

$$\begin{aligned} \alpha &= q_1^2 + q_2^2 + q_3^2 \\ \beta &= 2(p_1q_1 + p_2q_2 + p_3q_3) - q_4^2 \\ \gamma &= p_1^2 + p_2^2 + p_3^2 - 2p_4q_4 \\ \delta &= -p_4^2 \end{aligned} \tag{56.57}$$

At most, three solutions exist to the third-order equation. The speed of propagation estimate is given by the positive square root of the real positive solution. If there is more than one positive and real root, then the more ''physical'' estimate that fits the data is used. Once the speed of propagation is estimated by the constraint equation, the source location estimate can be given by $\mathbf{y} = \mathbf{p} + \hat{v}^2\mathbf{q}$. Compared with the LS method, the minimum required number of sensors in the CLS method can be further reduced by one.

56.2.2.4 Time-Delay Estimation Methods

Relative time-delay estimation is a classical problem in array signal processing, and plenty of algorithms have been proposed over the years. A collection of relative time-delay estimation papers

© 2005 by Chapman & Hall/CRC

has been put together by Carter [17], the pioneer in this field. The idea behind most relative time-delay estimation methods is based on maximizing the weighted cross-correlation between a pair of sensor data to extract an ML-type estimate. Denote the pair of sensor data as $x_p(t)$ and $x_q(t)$, where one is a delayed version of the other and corrupted by independent additive noise. The delay τ is estimated by the following:

$$\max_{\tau} \int_t w(t) x_p(t) x_q(t-\tau)\, \mathrm{d}t \tag{56.58}$$

where $w(t)$ is the weighting function appropriately chosen for the data type. Normally, $w(t)$ is a tapering function, i.e. maximum gain at the center of data and attenuation at the edge of data, which reduces the edge effect caused by the missing data at the edges (leading and trailing data in the pair) in a finite data window. This effect is especially magnified for low-frequency data in a short data window.

Note that, when the signal is relatively narrowband, an ambiguity in the correlation peak also occurs at every period of the signal. This is directly related to the grating lobes or spatial aliasing effect observed in the beampattern of an equally spaced array. Such a flaw can be avoided by limiting the search range of the delay to within the period of the signal, or equivalently limiting the angular/range search range in the spatial domain for direct localization or DOA estimation. However, for arrays with large element spacing, the ambiguity may be difficult to resolve without any prior information of the rough direction of the source, and a leading waveform may very well be confused with a trailing waveform by the estimator, hence leading to the ambiguous solution of the relative time delay. Normally, the relative time delay is performed independently between the reference sensor and every sensor other than the reference one. An erroneous location/DOA estimate results when one of the relative time-delay estimates is ambiguous. Nonetheless, as shown by Chen *et al.* [19], the parametric ML location/DOA estimator equivalently maximizes the summed cross-correlation of all pairs of sensors instead of separately maximizing for each pair; thus, the ML solution is less susceptible to the ambiguity problem.

For many applications, the relative time-delay may be very small comparing to the sampling rate of the data; thus, direct time-domain cross-correlation shown in (58) is usually performed after interpolation of the sensor data to achieve subsample estimate of the relative time-delay. An alternative approach is to perform cross-correlation in the frequency-domain, where the following is to be maximized

$$\max_{\tau} \int_{\omega} G(\omega) X_p(\omega) X_q^*(\omega)\, \mathrm{e}^{\mathrm{j}\omega\tau}\, \mathrm{d}\omega \tag{56.59}$$

where $G(\omega)$ is the frequency weighting function, $X(\omega)$ is the Fourier transform of $x(t)$, and * denotes the complex conjugate operation. The classical relative time-delay estimation method [17], the generalized cross-correlation (GCC) approach, conceived by Carter, is given in the above form where the choice of $G(\omega)$ leads to specific algorithms tailored for each application. One common choice of $G(\omega)$ is

$$G(\omega) = \frac{1}{\left| X_p(\omega) X_q^*(\omega) \right|}$$

which results in the PHAT algorithm that outperforms most algorithms in a nonreverberant environment. In a reverberant (multipath) environment, where the signal traverses via multiple paths other than the direct path, the aforementioned ambiguity issue becomes more severe, since now one

© 2005 by Chapman & Hall/CRC

sensor signal can be highly correlated with any nondirect path signal of the other sensor that arrives after a tiny delay. Depending on the frequency of the signal and the delay spread, this effect may be too distressing for some systems to work properly. More recent work on time-delay estimation focuses on robust estimators in the reverberant environment [31].

56.3 Array System Performance Analysis and Robust Design

In this section we compare the performance of several representative wideband source localization algorithms. This includes the ML parametric method in Equation (56.35), which we here refer to as the AML solution due to the finite length of the data,[1] the LS/CLS solution described in Equations (56.46) and (56.50), and the noncoherent wideband MUSIC algorithm proposed in by Tung *et al.* [32]. For performance comparison purposes, well-controlled computer simulations are used. The theoretical Cramér–Rao bound (CRB), the formulation of which is shown in the next section, is also compared with the performance of each algorithm. Then, the best algorithms are applied to the experimental data collected by wirelessly connected iPAQs with built-in microphones in section 56.4.2. The outstanding experimental results show that the sensor network technology is not just in the plain research stage, but is readily available for practical use.

56.3.1 Computer-Simulated Results for Acoustic Sources

Now, we consider some simulation examples and analysis for some acoustic sources. In principle, the methods used in the following can be applied to signals of other nature, e.g. seismic, but they will not be considered for the purpose of this section. In all cases, the sampling frequency is set to be 1 kHz. The speed of propagation is 345 m/s. A prerecorded tracked vehicle signal, with significant spectral content of about 50 Hz bandwidth centered about a dominant frequency at 100 Hz, is considered. For an arbitrary array of five sensors, we simulate the array data (with uniform SNR across the array) using this tracked vehicle signal with appropriate time delays. The data length $L = 200$ (which corresponds to 0.2 s), the FFT size $N = 256$ (with zero-padding), and all positive frequency bins are considered in the AML metric. To understand the fundamental properties of the AML algorithm, a normalized metric for a single source $J_{\mathrm{N}}(\mathbf{r}_s)$, which is a special case of Equation (56.35), is plotted in a range of possible locations. In this case

$$J_{\mathrm{N}}(\mathbf{r}_s) \equiv \frac{\sum_{k=1}^{N/2} \left|\mathbf{d}(k, \mathbf{r}_s)^{\mathrm{H}} \mathbf{X}(\omega_k)\right|^2}{R J_{\max}} \leq 1 \tag{56.60}$$

where $\mathbf{d}(k, \mathbf{r}_s)$ is the steering vector that steers to a particular position $\mathbf{r}_s$ and $J_{\max} = \sum_{k=1}^{N/2} [\sum_{p=1}^{R} |X_p(\omega_k)|]^2$, which is useful to verify estimated peak values. The source location can be estimated based on where $J_{\mathrm{N}}(\mathbf{r}_s)$ is maximized for a given set of locations. In Figure 56.6, $J_{\mathrm{N}}(\mathbf{r}_s)$ is evaluated at different near-field positions and plotted for a source inside the convex hull of the array (overlaid on top) under 20 dB SNR. A high peak shows up at the source location, thus indicating good estimation. When the source moves away from the array, the peak broadens and results in more range estimation error since it is more sensitive to noise. As depicted in Figure 56.7 (for the same setting), the image plot of $J_{\mathrm{N}}(\mathbf{r}_s)$ evaluated at different positions shows that the range estimation error is likely to occur in the source direction when the source moves away from the array. However, the angle estimation is not greatly impacted.

[1]For finite length data, the discrete Fourier transform (DFT) has a few artifacts. The circular shift property of the DFT introduces an edge effect problem for the actual linear time shift, and this edge effect is not negligible for a small block of data. To remove the edge effect, appropriate zero padding can be applied. However, it is also known that zero padding destoys the orthogonality of the DFT, which makes the noise spectrum appear correlated across frequency. As a result, there does not exist an exact ML solution for data of finite length.

© 2005 by Chapman & Hall/CRC

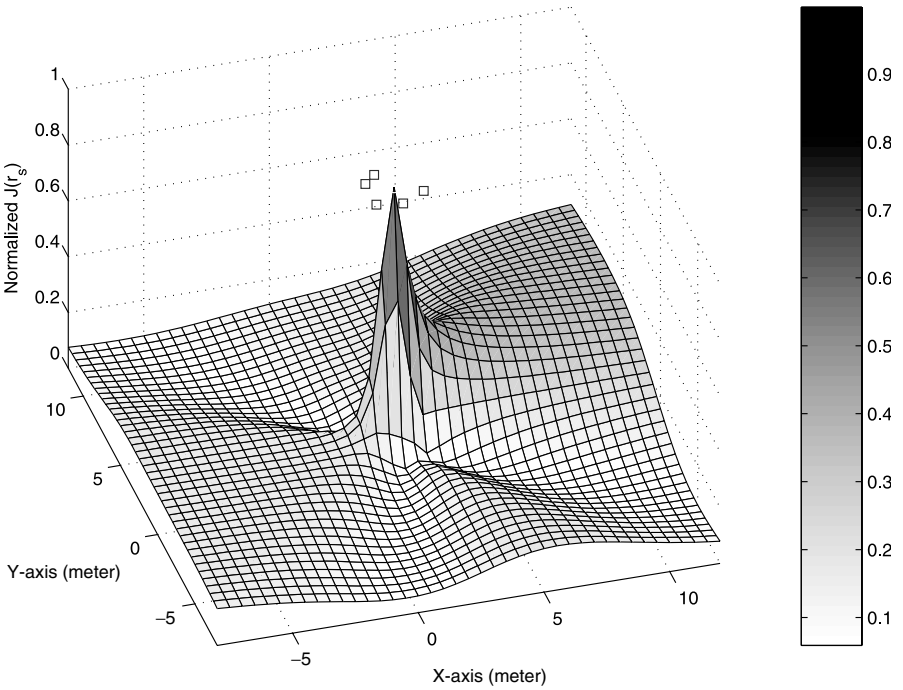


Figure 56.6. The three-dimensional plot of $J_N(\mathbf{r}_s)$ for a source inside the convex hull of the array. Square: sensor locations.

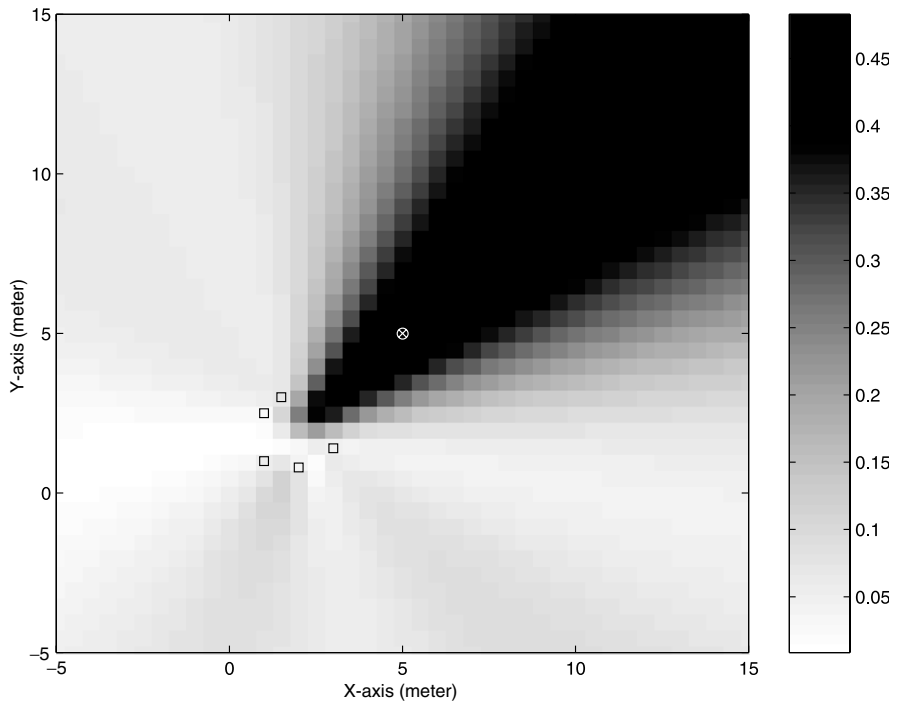


Figure 56.7. The image plot of $J_N(\mathbf{r}_s)$ for a distant source. Square: sensor locations; circle: actual source location; $\times$: source location estimate.

© 2005 by Chapman & Hall/CRC

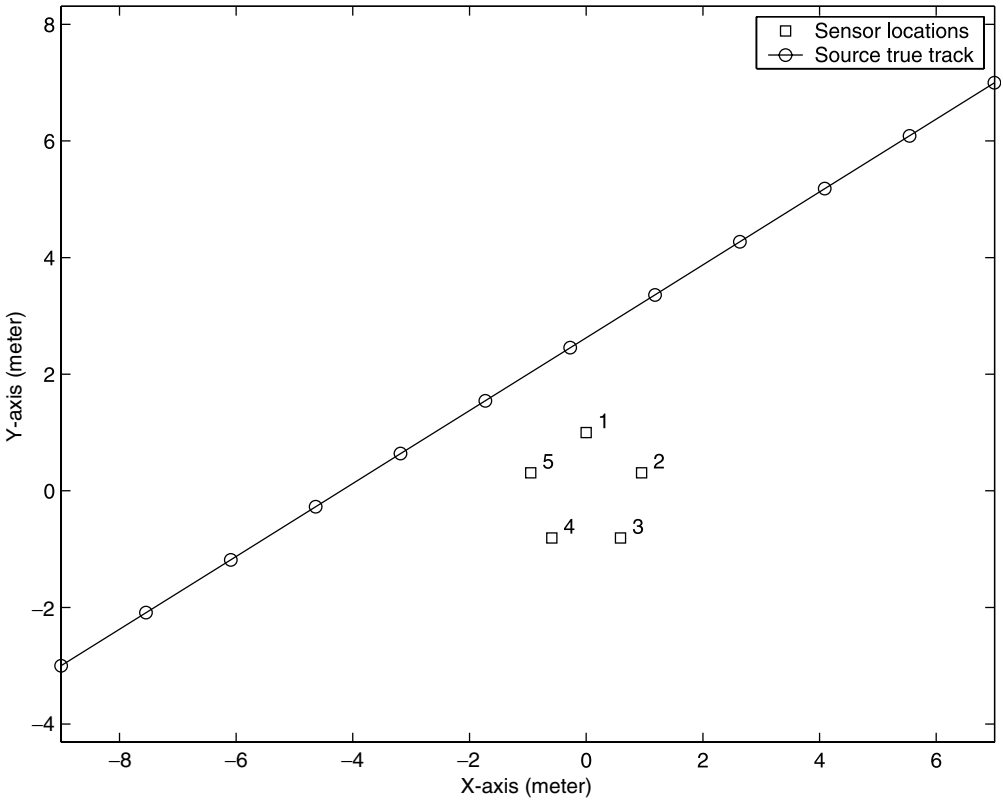


Figure 56.8. Single traveling source scenario.

In the following, we consider a single traveling source scenario for a circular array (uniformly spaced on the circumference), as depicted in Figure 56.8. In this case we consider the spatial loss that is a function of the distance from the source location to each sensor location; thus, the SNR is no longer uniform. For each frame of $L = 200$ samples, we simulate the array data for one of the 12 source locations. We apply the AML method without weighting the data according to the SNR and estimate the source location at each frame. A two-dimensional polar grid-point system is being used, where the angle is uniformly sampled and the range is uniformly sampled on a log-scale (to be physically meaningful). This two-dimensional grid-point search provides the initial estimate of the direct search simplex method in Matlab's Optimization Toolbox; thus, a global maximum is guaranteed. In addition, we apply the LS method [29] and the wideband MUSIC method [32] and compute the CRB using Equation (56.75) for comparison purposes. Generally speaking, the LS is the most computationally efficient, then the AML and the wideband MUSIC, but the detailed complexities of these methods are not compared here. The wideband MUSIC divides the 200 samples into four subintervals of 50 samples each for the computation of the sample correlation matrix and uses the same grid-point search and direct search method as the AML case. As depicted in Figure 56.9 ((a) for $R = 5$ and (b) for $R = 7$), both the AML and LS methods approach the CRB when the source is near the array (high SNR and short range), but the AML outperforms the LS method. However, the wideband MUSIC yields much worse estimates than those of the LS and AML methods, especially when the source is far from the array. The wideband MUSIC requires a large number of samples for good performance, but in this case L is limited by the moving source. Note that when the source is far from the array (low SNR and long range) the CRB is too idealistic, since it does not take into account other factors such as the edge effect or correlated noise across frequency. Thus, the reader should not get the impression that the AML performs poorly in this region.

© 2005 by Chapman & Hall/CRC

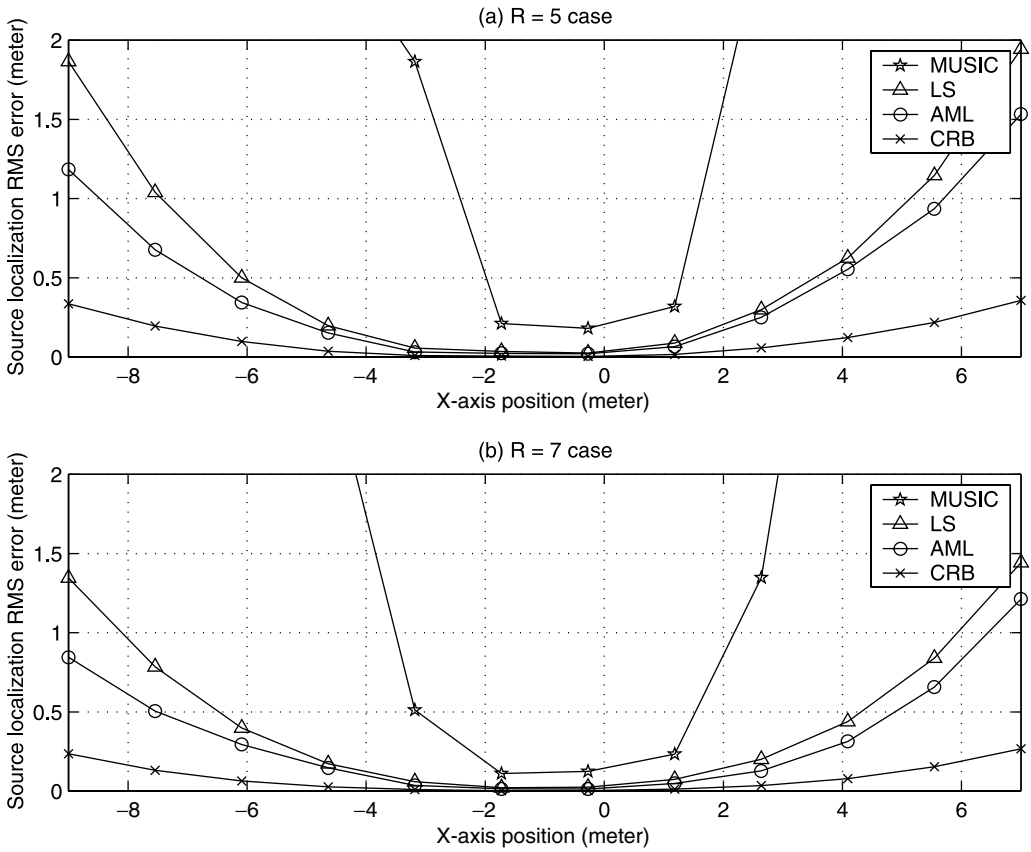


Figure 56.9. Source localization performance comparison for known speed of propagation case: (a) $R = 5$; (b) $R = 7$.

56.3.2 CRB for Source Localization

The CRB is most often used as a theoretical lower bound for any unbiased estimator [33]. Normally, the result of an algorithm is compared with the CRB for performance merit. However, the CRB provides not just a lower bound, it also provides very valuable information regarding the nature of the problem. By analyzing the CRB, we can find many insights to the problem at hand and it can help the design of the array for optimum performance. In the following we provide the formulation of two CRBs based on time-delay error and SNR. The CRB based on time-delay error can be used to analyze and compare the time-delay-based algorithms, and the CRB based on SNR, which is more generalized, can be used to analyze and compare all algorithms.

56.3.2.1 CRB Based on Time-Delay Error

In this section we derive the CRB for source localization and speed of propagation estimations with respect to the relative time-delay estimation error. Denote the unknown parameter vector by $\mathbf{\Theta} = \left[\mathbf{r}_s^{\mathrm{T}}, \nu\right]^{\mathrm{T}}$. The measured relative time-delay vector can be modeled by

$$\widehat{\mathbf{t}} = \mathbf{t}(\mathbf{\Theta}) + \xi \tag{56.61}$$

where $\mathbf{t} = [t_{21}, \ldots, t_{R1}]^{\mathrm{T}}$ is the true relative time-delay vector and ξ is the estimation error vector, which is assumed to be zero mean white Gaussian distributed with variance σ^2. Define the gradient matrix

© 2005 by Chapman & Hall/CRC

$\mathbf{H} = \partial \mathbf{t}/\partial \mathbf{\Theta}^{\mathrm{T}} = \left[\partial \mathbf{t}/\partial \mathbf{r}_s^{\mathrm{T}},\ \partial \mathbf{t}/\partial v\right] = (1/v)[\mathbf{B},\ -\mathbf{t}]$, where $\mathbf{B} = [\mathbf{u}_2 - \mathbf{u}_1, \ldots, \mathbf{u}_R - \mathbf{u}_1]^{\mathrm{T}}$ and the unit vector $\mathbf{u}_r = (\mathbf{r}_s - \mathbf{r}_r)/\|\mathbf{r}_s - \mathbf{r}_r\|$ indicates the direction of the source from the rth sensor. The Fisher information matrix [33] is then given by

$$\begin{aligned}\mathbf{F}_{\mathbf{r}_s} &= \mathbf{H}^{\mathrm{T}}\mathbf{R}_{\xi}^{-1}\mathbf{H} = \left(1/\sigma^2\right)\mathbf{H}^{\mathrm{T}}\mathbf{H} \\ &= \left(1/\sigma^2 v^2\right)\begin{bmatrix} \mathbf{B}^{\mathrm{T}}\mathbf{B} & -\mathbf{B}^{\mathrm{T}}\mathbf{t} \\ -\mathbf{t}^{\mathrm{T}}\mathbf{B} & \|\mathbf{t}\|^2 \end{bmatrix}\end{aligned} \tag{56.62}$$

The theoretical lower bound of the variance of $\mathbf{r}_s$ is given by the diagonal elements of the leading 3×3 submatrix of the inverse Fisher information matrix

$$\left[\mathbf{F}_{\mathbf{r}_s}^{-1}\right]_{11:33} = \sigma^2 v^2\left[\mathbf{B}^{\mathrm{T}}\mathbf{P}_{\mathbf{t}}^{\perp}\mathbf{B}\right]^{-1} \tag{56.63}$$

where $\mathbf{P}_{\mathbf{t}} = \mathbf{t}\mathbf{t}^{\mathrm{T}}/\|\mathbf{t}\|^2$ is the orthogonal projection matrix of $\mathbf{t}$ and $\mathbf{P}_{\mathbf{t}}^{\perp} = \mathbf{I} - \mathbf{P}_{\mathbf{t}}$. The variance bound of the distance d between the estimated source location and the actual location is given by $\sigma_d^2 \geq \text{trace}\,[\mathbf{F}_{\mathbf{r}_s}^{-1}]_{11:33}$. The variance bound of the speed of propagation is given by

$$\sigma_v^2 \geq \left[\mathbf{F}_{\mathbf{r}_s}^{-1}\right]_{44} = \sigma^2 v^2\left\|\mathbf{P}_{\mathbf{B}}^{\perp}\mathbf{t}\right\|^{-2} \tag{56.64}$$

where $\mathbf{P}_{\mathbf{B}} = \mathbf{B}\left(\mathbf{B}^{\mathrm{T}}\mathbf{B}\right)^{-1}\mathbf{B}^{\mathrm{T}}$ is the orthogonal projection matrix of $\mathbf{B}$ and $\mathbf{P}_{\mathbf{B}}^{\perp} = \mathbf{I} - \mathbf{P}_{\mathbf{B}}$.

The CRB analysis shows theoretically that the variance of source location estimation grows linearly with the relative time-delay estimation variance. The standard deviations of both the location and speed of propagation estimation errors grow linearly with the value of the speed of propagation. Note that, when the speed of propagation is known, the source location estimation variance bound becomes $\sigma_d^2 \geq \sigma^2 v^2\ \text{trace}\,[\left(\mathbf{B}^{\mathrm{T}}\mathbf{B}\right)^{-1}]$, which is always smaller than that of the unknown speed of propagation case. Define the *array matrix* by $\mathbf{A}_{\mathbf{r}_s} = \mathbf{B}^{\mathrm{T}}\mathbf{B}$. It provides a measure of geometric relations between the source and the sensor array. Poor array geometry may lead to degeneration in the rank of matrix $\mathbf{A}_{\mathbf{r}_s}$, thus resulting in a large estimation error bound.

By transforming to the polar coordinate system in the two-dimensional case, the CRB for the source range and DOA can also be given. Denote $r_s = \sqrt{x_s^2 + y_s^2}$ as the source range from a reference position such as the array centroid. The DOA can be given by $\phi_s = \tan^{-1}(x_s/y_s)$ with respect to the Y-axis. The time delay from the source to the rth sensor is given by $t_r = \sqrt{r_s^2 + r_r^2 - 2r_s r_r\ \cos(\phi_s - \phi_r)}/v$, where (r_r, ϕ_r) is the location of the rth sensor in the polar coordinate system. The gradient with respect to the source range is given by

$$\frac{\partial t_{r1}}{\partial r_s} = \frac{1}{v}\left[\frac{r_s - r_r\ \cos(\phi_s - \phi_r)}{\|\mathbf{r}_s - \mathbf{r}_r\|} - \frac{r_s - r_1\ \cos(\phi_s - \phi_1)}{\|\mathbf{r}_s - \mathbf{r}_1\|}\right] \tag{56.65}$$

and the gradient with respect to the DOA is given by

$$\frac{\partial t_{r1}}{\partial \phi_s} = \frac{1}{v}\left[\frac{r_s r_r\ \sin(\phi_s - \phi_r)}{\|\mathbf{r}_s - \mathbf{r}_r\|} - \frac{r_s r_1\ \sin(\phi_s - \phi_1)}{\|\mathbf{r}_s - \mathbf{r}_1\|}\right] \tag{56.66}$$

Define $\mathbf{B}_{\text{pol}} = v[\partial \mathbf{t}/\partial r_s,\ \partial \mathbf{t}/\partial \phi_s]$. The polar array matrix is then defined by $\mathbf{A}^{\text{pol}} = \mathbf{B}_{\text{pol}}^{\mathrm{T}}\mathbf{B}_{\text{pol}}$. The leading 2×2 submatrix of the inverse polar Fisher information matrix can be given by

$$\left[\mathbf{F}_{(r_s,\phi_s)}^{-1}\right]_{11:22} = \sigma^2 v^2\left[\mathbf{B}_{\text{pol}}^{\mathrm{T}}\mathbf{P}_{\mathbf{t}}^{\perp}\mathbf{B}_{\text{pol}}\right]^{-1} \tag{56.67}$$

© 2005 by Chapman & Hall/CRC

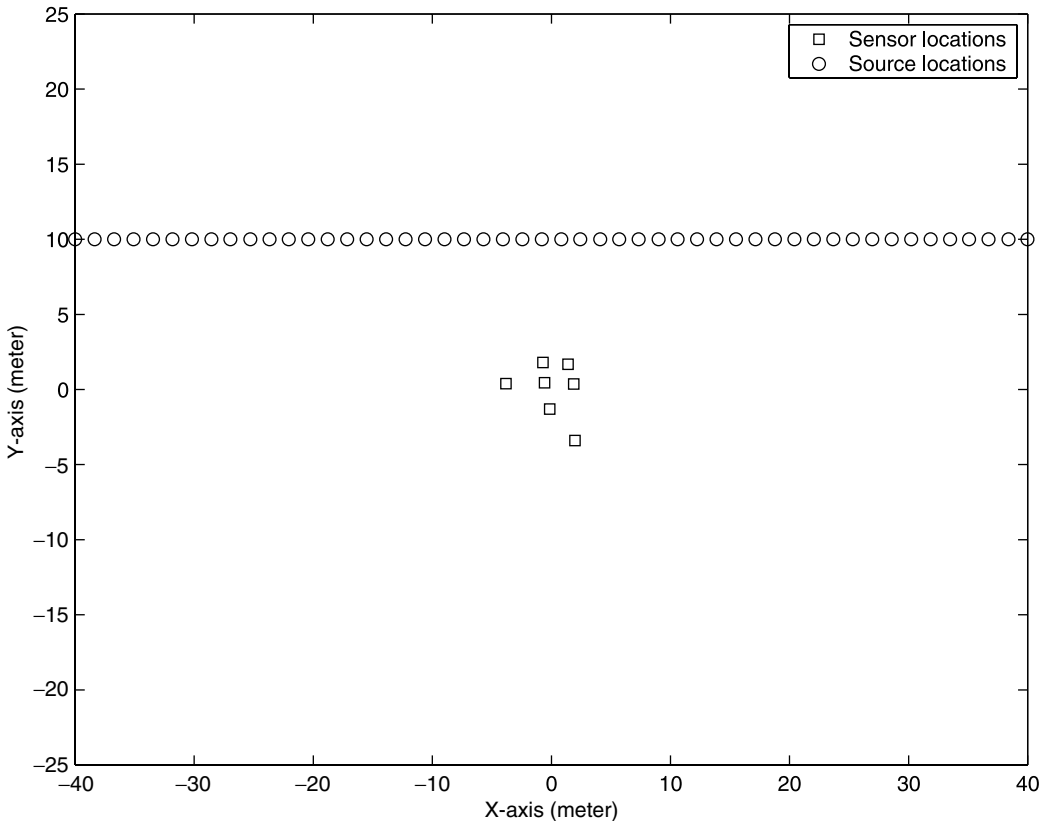


Figure 56.10. Traveling source scenario.

Then, the lower bound of range estimation variance can be given by $\sigma_{r_s}^2 \geq [\mathbf{F}_{(r_s,\phi_s)}^{-1}]_{11}$ and the lower bound of DOA estimation variance can be given by $\sigma_{\phi_s}^2 \geq [\mathbf{F}_{(r_s,\phi_s)}^{-1}]_{22}$.

To evaluate the performance of the LS and CLS source localization algorithms by comparing with the theoretical Cramér–Rao lower bound, the simulated scenario depicted in Figure 56.10 is used. A randomly distributed array of seven sensors is used to collect the signal generated by a source moving in a straight line. By perturbing the actual time delays by a white Gaussian noise with zero mean and standard deviation of $\sigma_{\text{td}} = 10\,\mu\text{s}$, the LS and CLS algorithms are applied to estimate the source location at each time frame. As depicted in Figure 56.11, the CLS yields much better range and DOA estimates than the LS on the average of 10,000 random realizations. The CLS estimates are also very close to the corresponding CRB.

56.3.2.2 CRB Based on SNR

Most of the derivations of the CRB for wideband source localization found in the literature are in terms of relative time-delay estimation error, as shown in the previous section. In this section we derive a more general CRB directly from the signal model. By developing a theoretical lower bound in terms of signal characteristics and array geometry, we not only bypass the involvement of the intermediate time-delay estimator, but also offer useful insights to the physics of the problem.

We consider the following three cases: known signal and known speed of propagation; known signal but unknown speed of propagation; and known speed of propagation but unknown signal. The comparison of the three conditions provides a sensitivity analysis that explains the fundamental differences of different problems, e.g. unknown speed of propagation for seismic sensing and known signal for radar applications. The case of unknown signal and unknown speed of propagation is not too

© 2005 by Chapman & Hall/CRC

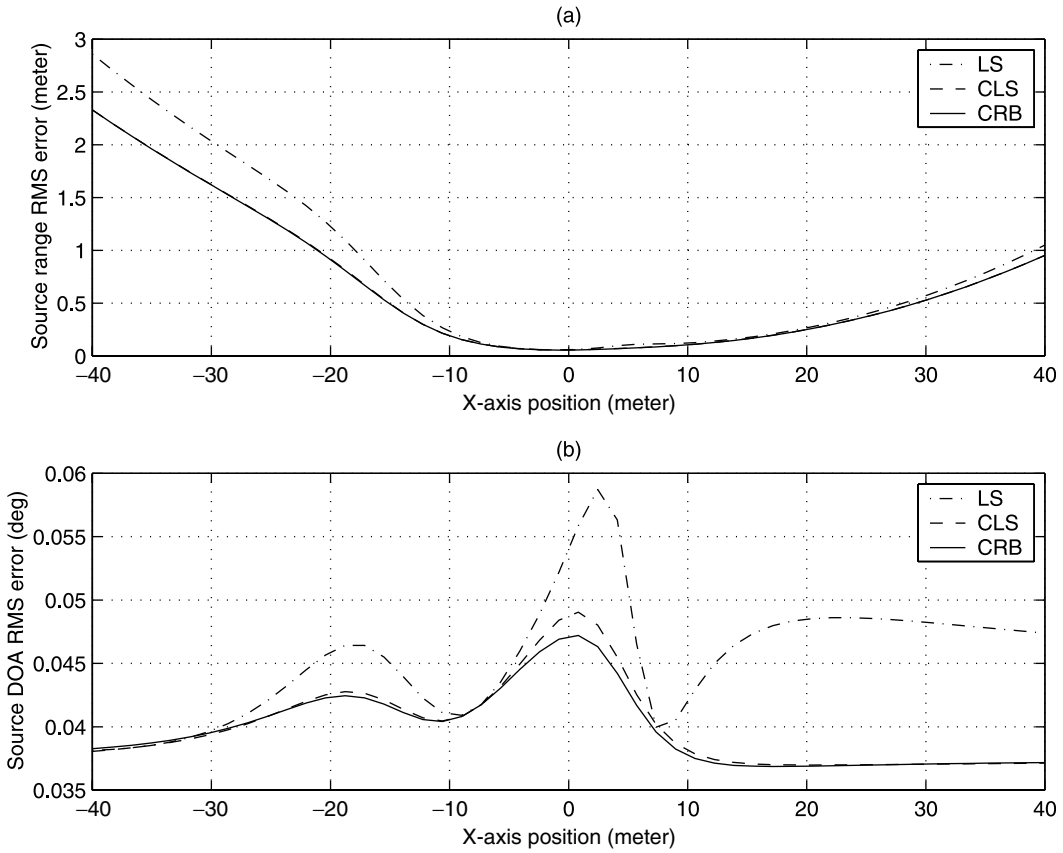


Figure 56.11. LS, CLS, and CRB location root-mean-square error at various source locations ($\sigma_{\text{td}} = 10\mu\text{s}$).

different from the case of unknown signal and known speed of propagation; thus, it is not considered. For all cases, we can construct the Fisher information matrix [33] from the signal model by

$$\mathbf{F} = 2\text{Re}\left[\mathbf{H}^{\text{H}}\mathbf{R}_{\xi}^{-1}\mathbf{H}\right] = \left(2/L\sigma^2\right)\text{Re}\left[\mathbf{H}^{\text{H}}\mathbf{H}\right] \tag{56.68}$$

where $\mathbf{H} = \partial\mathbf{G}/\partial\mathbf{r}_s^{\text{T}}$ for the first case, assuming $\mathbf{r}_s$ is the only unknown in the single source case. In this case, $\mathbf{F}_{\mathbf{r}_s} = \zeta\mathbf{A}$, where $\zeta = \left(2/L\sigma^2\nu^2\right)\sum_{k=1}^{N/2}\left(2\pi k\left|S_0(k)\right|/N\right)^2$ is the scale factor that is proportional to the total power in the derivative of the source signal,

$$\mathbf{A} = \sum_{p=1}^{R} a_p^2\mathbf{u}_p\mathbf{u}_p^{\text{T}} \tag{56.69}$$

is the *array matrix*, and $\mathbf{u}_p = \left(\mathbf{r}_s - \mathbf{r}_p\right)/\left\|\mathbf{r}_s - \mathbf{r}_p\right\|$ is the unit vector indicating the direction of the source from the pth sensor. The $\mathbf{A}$ matrix provides a measure of geometric relations between the source and the sensor array. Poor array geometry may lead to degeneration in the rank of matrix $\mathbf{A}$. It is clear from the scale factor ζ, as will be shown later, that the performance does not depend solely on the SNR, but also on the signal bandwidth and spectral density. Thus, source localization performance is better for signals with more energy in the high frequencies.

© 2005 by Chapman & Hall/CRC

For the second case, when the speed of propagation is also unknown, i.e. $\boldsymbol{\Theta} = [\mathbf{r}_s^{\mathrm{T}}, v]^{\mathrm{T}}$, the $\mathbf{H}$ matrix for this case is given by $\mathbf{H} = [\partial\mathbf{G}/\partial\mathbf{r}_s^{\mathrm{T}}, \partial\mathbf{G}/\partial v]$. The Fisher information block matrix is given by

$$\mathbf{F}_{\mathbf{r}_s, v} = \zeta \begin{bmatrix} \mathbf{A} & -\mathbf{U}\mathbf{A}_a\mathbf{t} \\ -\mathbf{t}^{\mathrm{T}}\mathbf{A}_a\mathbf{U}^{\mathrm{T}} & \mathbf{t}^{\mathrm{T}}\mathbf{A}_a\mathbf{t} \end{bmatrix} \tag{56.70}$$

where $\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_R]$, $\mathbf{A}_a = \mathrm{diag}([a_1^2, \ldots, a_R^2])$, and $\mathbf{t} = [t_1, \ldots, t_R]^{\mathrm{T}}$. By applying the well-known block matrix inversion lemma, the leading $D \times D$ submatrix of the inverse Fisher information block matrix can be given by

$$\left[\mathbf{F}_{\mathbf{r}_s, v}^{-1}\right]_{11:DD} = \frac{1}{\zeta}(\mathbf{A} - \mathbf{Z}_v)^{-1} \tag{56.71}$$

where the *penalty matrix* due to unknown speed of propagation is defined by $\mathbf{Z}_v = (1/\mathbf{t}^{\mathrm{T}}\mathbf{A}_a\mathbf{t})\mathbf{U}\mathbf{A}_a\mathbf{t}\mathbf{t}^{\mathrm{T}}\mathbf{A}_a\mathbf{U}^{\mathrm{T}}$. The matrix $\mathbf{Z}_v$ is nonnegative definite; therefore, the source localization error of the unknown speed of propagation case is always larger than that of the known case.

For the third case, when the source signal is also unknown, i.e. $\boldsymbol{\Theta} = [\mathbf{r}_s^{\mathrm{T}}, |\mathbf{S}_0|^{\mathrm{T}}, \boldsymbol{\Phi}_0^{\mathrm{T}}]^{\mathrm{T}}$, the $\mathbf{H}$ matrix is given by

$$\mathbf{H} = \left[\frac{\partial\mathbf{G}}{\partial\mathbf{r}_s^{\mathrm{T}}}, \frac{\partial\mathbf{G}}{\partial|\mathbf{S}_0|^{\mathrm{T}}}, \frac{\partial\mathbf{G}}{\partial\boldsymbol{\Phi}_0^{\mathrm{T}}}\right]$$

where $\mathbf{S}_0 = [S_0(1), \ldots, S_0(N/2)]^{\mathrm{T}}$, and $|\mathbf{S}_0|$ and $\boldsymbol{\Phi}_0$ are the magnitude and phase part of $\mathbf{S}_0$ respectively. The Fisher information matrix can then be explicitly given by

$$\mathbf{F}_{\mathbf{r}_s, \mathbf{S}_0} = \begin{bmatrix} \zeta\mathbf{A} & \mathbf{B} \\ \mathbf{B}^{\mathrm{T}} & \mathbf{D} \end{bmatrix} \tag{56.72}$$

where $\mathbf{B}$ and $\mathbf{D}$ are not explicitly given since they are not needed in the final expression. By applying the block matrix inversion lemma, the leading $D \times D$ submatrix of the inverse Fisher information block matrix can be given by

$$\left[\mathbf{F}_{\mathbf{r}_s, \mathbf{S}_0}^{-1}\right]_{11:DD} = \frac{1}{\zeta}(\mathbf{A} - \mathbf{Z}_{\mathbf{S}_0})^{-1} \tag{56.73}$$

where the penalty matrix due to unknown source signal is defined by

$$\mathbf{Z}_{\mathbf{S}_0} = \frac{1}{\sum_{p=1}^{R} a_p^2}\left(\sum_{p=1}^{R} a_p^2\mathbf{u}_p\right)\left(\sum_{p=1}^{R} a_p^2\mathbf{u}_p\right)^{\mathrm{T}} \tag{56.74}$$

The CRB with unknown source signal is always larger than that with known source signal, as discussed below. This can easily be shown, since the penalty matrix $\mathbf{Z}_{\mathbf{S}_0}$ is nonnegative definite. The $\mathbf{Z}_{\mathbf{S}_0}$ matrix acts as a penalty term, since it is the average of the square of weighted $\mathbf{u}_p$ vectors. The estimation variance is larger when the source is far away, since the $\mathbf{u}_p$ vectors are similar in direction and generate a larger penalty matrix, i.e. $\mathbf{u}_p$ vectors add up. When the source is inside the convex hull of the sensor array, the estimation variance is smaller, since $\mathbf{Z}_{\mathbf{S}_0}$ approaches the zero matrix, i.e. $\mathbf{u}_p$ vectors cancel

© 2005 by Chapman & Hall/CRC

each other. For the two-dimensional case, the CRB for the distance error of the estimated location $[\hat{x}_s, \hat{y}_s]^{\mathrm{T}}$ from the true source location can be given by

$$\sigma_d^2 = \sigma_{x_s}^2 + \sigma_{y_s}^2 \geq \left[\mathbf{F}_{\mathbf{r}_s,\mathbf{S}_0}^{-1}\right]_{11} + \left[\mathbf{F}_{\mathbf{r}_s,\mathbf{S}_0}^{-1}\right]_{22} \tag{56.75}$$

where $d^2 = (\hat{x}_s - x_s)^2 + (\hat{y}_s - y_s)^2$. By further expanding the parameter space, the CRB for multiple source localizations can also be derived, but its analytical expression is much more complicated. Note that, when both the source signal and sensor gains are unknown, it is not possible to determine the values of the source signal and the sensor gains (they can only be estimated up to a scaled constant).

By transforming to the polar coordinate system in the two-dimensional case, the CRB for the source range and DOA can also be given. Denote $r_s = \sqrt{x_s^2 + y_s^2}$ as the source range from a reference position such as the array centroid. The DOA can be given by $\phi_s = \tan^{-1}(x_s/y_s)$ with respect to the *Y*-axis. The time delay from the source to the pth sensor is then given by $t_p = \sqrt{r_s^2 + r_p^2 - 2r_s r_p \cos(\phi_s - \phi_p)}/v$, where (r_p, ϕ_p) is the polar position of the pth sensor. We can form a polar array matrix $\mathbf{A}^{\mathrm{pol}} = \sum_{p=1}^{R} a_p^2 \mathbf{v}_p \mathbf{v}_p^{\mathrm{T}}$, where

$$\mathbf{v}_p = \left[\frac{r_s - r_p \cos(\phi_s - \phi_p)}{\|\mathbf{r}_s - \mathbf{r}_p\|}, \frac{r_s r_p \sin(\phi_s - \phi_p)}{\|\mathbf{r}_s - \mathbf{r}_p\|}\right]^{\mathrm{T}} \tag{56.76}$$

Similarly, the polar penalty matrix due to the unknown source signal can be given by

$$\mathbf{Z}_{\mathbf{S}_0}^{\mathrm{pol}} = \frac{1}{\sum_{p=1}^{R} a_p^2}\left(\sum_{p=1}^{R} a_p^2 \mathbf{v}_p\right)\left(\sum_{p=1}^{R} a_p^2 \mathbf{v}_p\right)^{\mathrm{T}} \tag{56.77}$$

The leading 2×2 submatrix of the inverse polar Fisher information block matrix can be given by

$$\left[\mathbf{F}_{(r_s,\phi_s),\mathbf{S}_0}^{-1}\right]_{11:22} = \frac{1}{\zeta}(\mathbf{A}^{\mathrm{pol}} - \mathbf{Z}_{\mathbf{S}_0}^{\mathrm{pol}})^{-1} \tag{56.78}$$

Then, the lower bound of range estimation variance can be given by $\sigma_{r_s}^2 \geq [\mathbf{F}_{(r_s,\phi_s),\mathbf{S}_0}^{-1}]_{11}$ and the lower bound of DOA estimation variance can be given by $\sigma_{\phi_s}^2 \geq [\mathbf{F}_{(r_s,\phi_s),\mathbf{S}_0}^{-1}]_{22}$. The polar array matrix shows that a good DOA estimate can be obtained at the broadside of a linear array and a poor DOA estimate results at the endfire of a linear array. A two-dimensional array is required for better range and DOA estimations, e.g. a circular array.

To compare the theoretical performance of source localization under different conditions, we compare the CRB for the known source signal and speed of propagation, unknown speed of propagation, and unknown source signal cases using seven sensors for the same single traveling source scenario (shown in Figure 56.8). As depicted in Figure 56.12, the unknown source signal is shown to have a much more significant degrading effect than the unknown speed of propagation in source location estimation. However, these parameters are not significant in the DOA estimations. Furthermore, the theoretical performance of the estimator is analyzed for different signal characteristics. For the same setting as the one shown in Figure 56.7, we replace the tracked vehicle signal by a Gaussian signal after a band-pass filter (with fixed SNR of 20 dB). The center frequency of the source signal is first set to 250 Hz and then we let the bandwidth of this signal vary from 50 to 450 Hz. As shown in Figure 56.13(a), the theoretical performance improves as the bandwidth increases. Then, for a fixed bandwidth of 100 Hz, we vary the center frequency from 50 to 450 Hz and observe the performance

© 2005 by Chapman & Hall/CRC

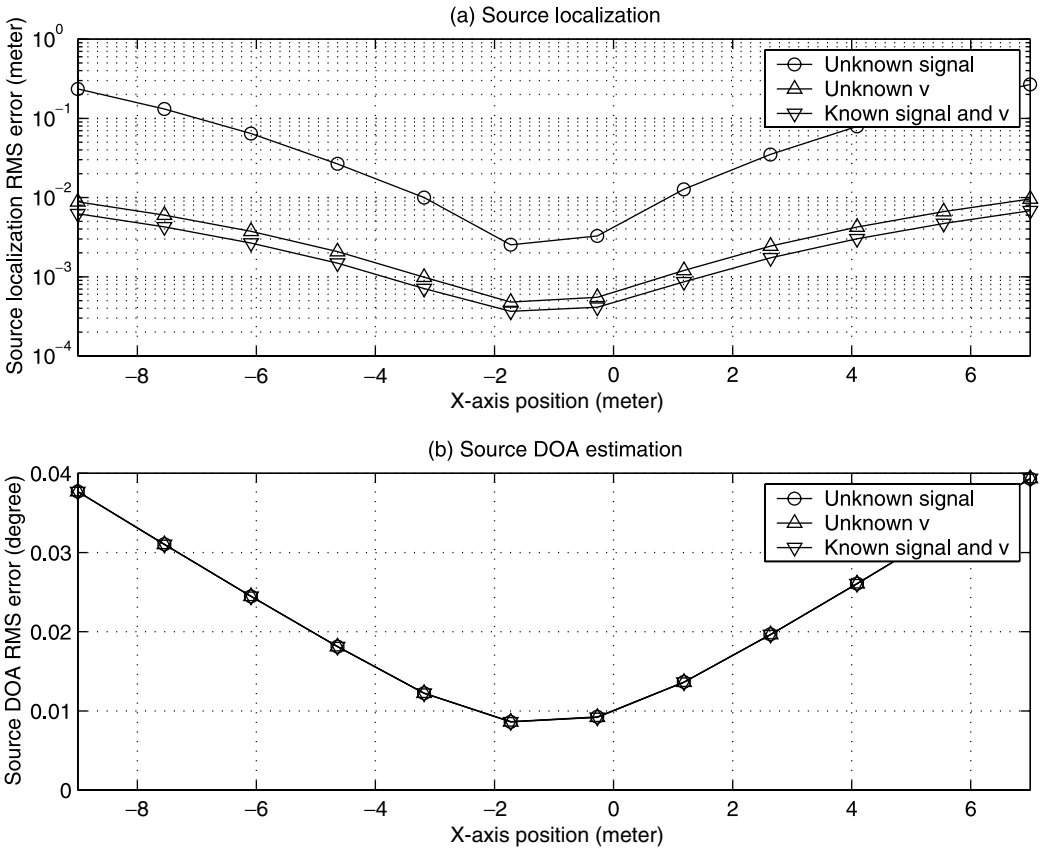


Figure 56.12. CRB comparison for the traveling source scenario ($R = 7$): (a) localization bound; (b) DOA bound.

improvement for higher center frequency in Figure 56.13(b). The theoretical performance analysis applies to the AML estimator, since the AML estimator approaches the CRB asymptotically.

56.3.3 Robust Array Design

For a coherent array beamforming operation, it is well known that the system needs to be well calibrated and that many realistic system perturbations may cause the array performance to degrade significantly [34]. Various robust array designs have been proposed [35,36]. Recently, a new class of robust array designs [37–39] based on the use of semi-definite programming (SDP) optimization methodology [40,41] and efficient Matlab programming code [42] has been proposed.

From calculus, it is clear that, for a polynomial function of degree two, their robust array designs have either a minimum or a maximum, and the local minimum or maximum is the global minimum or maximum, since the function is convex. Certainly, even for an arbitrary function of a single or vector variable, a local extremum solution does not guarantee a global extremum solution. However, if the optimizing function is convex and the domain is also convex, then a local extremum solution guarantees a global extremum solution. An SDP is a special class of tractable convex optimization problem in which every stationary point is also a global optimizing point, the global solution can be computed in small number of iterations, necessary and sufficient conditions for optimality can be obtained readily, and a well-developed duality theory exists that can determine whether a feasible solution exists (for the given constraints).

© 2005 by Chapman & Hall/CRC

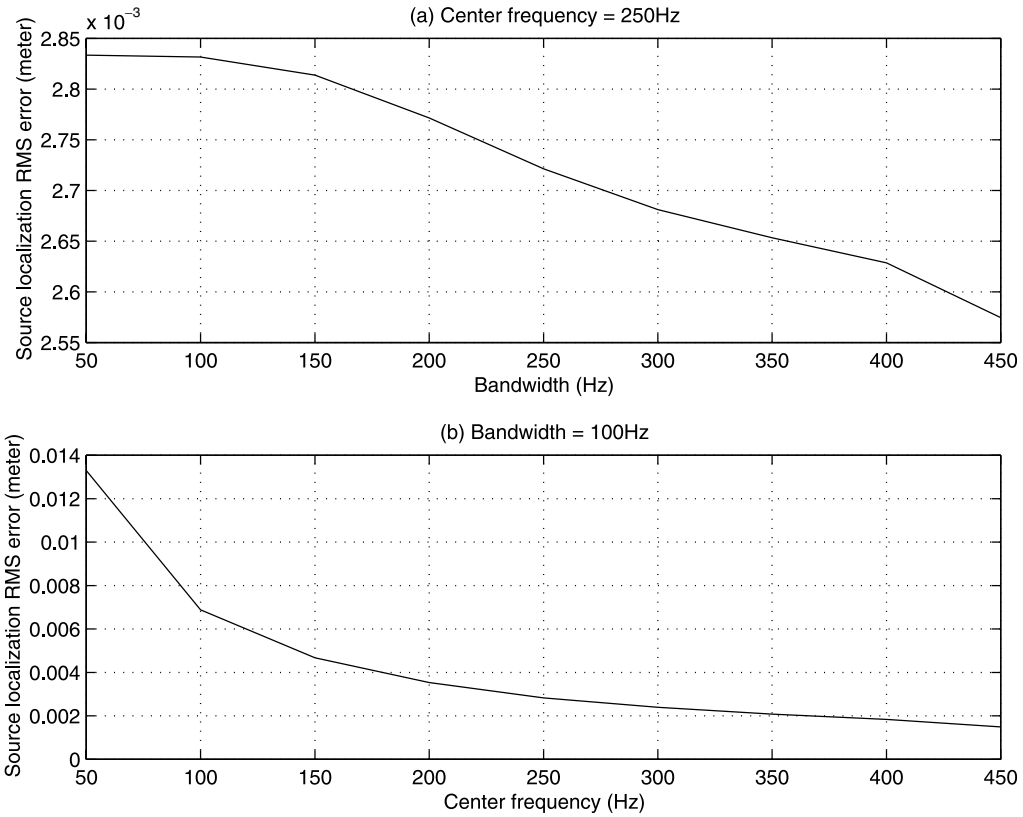


Figure 56.13. CRB comparison for different signal characteristics: (a) bandwidth; (b) center frequency.

The general SDP problem [38] can be defined as follows. A linear matrix inequality for the vector variable $x = [x_1, \ldots, x_m] \in R^m$ has the form of

$$F(x) = F_0 + \sum_{i=1}^{m} x_i F_i \geq 0 \tag{56.79}$$

where $F_i = F_i^{\mathrm{T}} \in R^{n \times n}$, $i = 0, \ldots, m$, are some specified matrices. $F(x)$ is an affine function of x and the feasible set is convex. Then, the general SDP problem is the convex optimization problem of

$$\mathrm{Min}\,\{c^{\mathrm{T}}x\}, \quad \text{subject to } F(x) \geq 0 \tag{56.80}$$

for any vector $c \in R^n$. A special case of the SDP problem called the *Second Order Cone Problem*, (SOCP) has the form of

$$\mathrm{Min}\{c^{\mathrm{T}}x\}, \quad \text{subject to } ||A_i x + b_i|| \leq c_i^{\mathrm{T}} + d_i, \; i = 1, \ldots, N \tag{56.81}$$

where $A_i \in C^n$, $b_i \in C^N$, and $D_i \in R$. Many array-processing design problems can be formulated and solved as a second-order cone program [38].

We show a simple robust array design problem [43] based on the formulation and solution as an SOCP (but omitting details). Consider a uniformly spaced circle array of 15 elements as denoted by the

© 2005 by Chapman & Hall/CRC

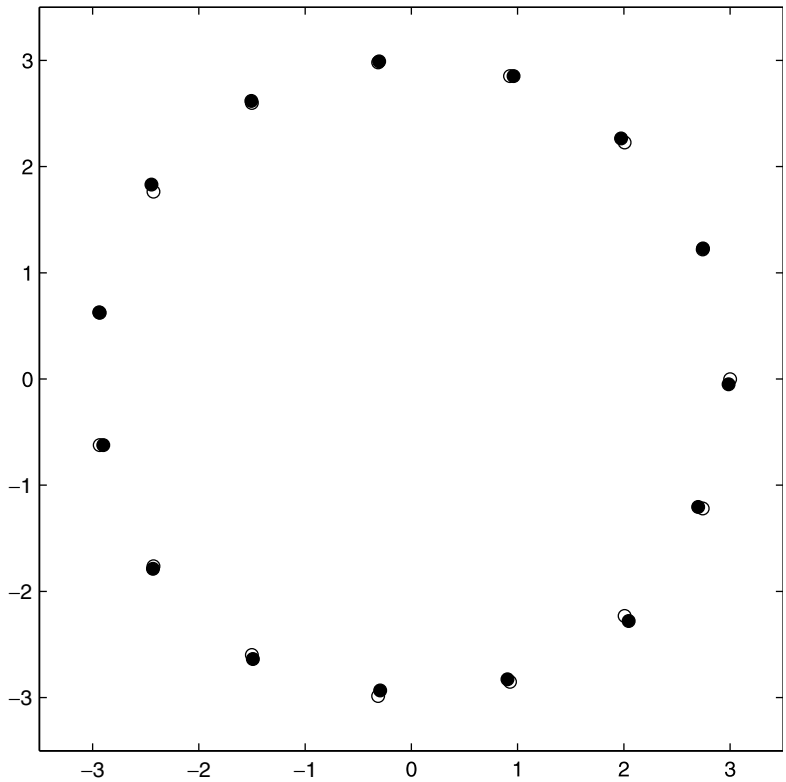


Figure 56.14. Placement of 15 uniformly spaced antenna elements. (a) ideal circular array (open circles); (b) perturbed array (black circles).

open circles in Figure 56.14. The array magnitude response at the desired angle of 0° is constrained to have unity value with all the sidelobe peaks to be equal or below the 0.1 value (i.e. 20 dB below the desired response). The resulting array magnitude response using standard beamforming weight design is shown in Figure 56.15(a). This response meets the desired constraints. However, for the weights designed for the ideal circular array but applied to a slightly perturbed array as denoted by the black dots in Figure 56.14, the new array response is given by Figure 56.15(b), which is probably not acceptable with the large sidelobe values near 0°. Now, using an SOCP array design, we note in Figure 56.16(a), the new sidelobe values for the ideal circular array are indeed slightly worse than those in Figure 56.15(a). But, the array response for the perturbed array shown in Figure 56.16(b) is only slightly worse than that for the ideal circular array response of Figure 56.16(a), and is certainly much more robust than that in Figure 56.15(b). This simple example shows the importance of robust array design to meeting practical system imperfections.

56.4 Implementations of Two Wideband Beamforming Systems

56.4.1 Implementation of a Radar Wideband Beamformer Using a Subband Approach

Modern wideband radar space–time adaptive processing problems may need to consider hundreds of sensor channels to form dozens of beams at high sampling rate [44–46]. Given the wideband nature of the radar waveform, subband domain approach is a known attractive method to decompose the

© 2005 by Chapman & Hall/CRC

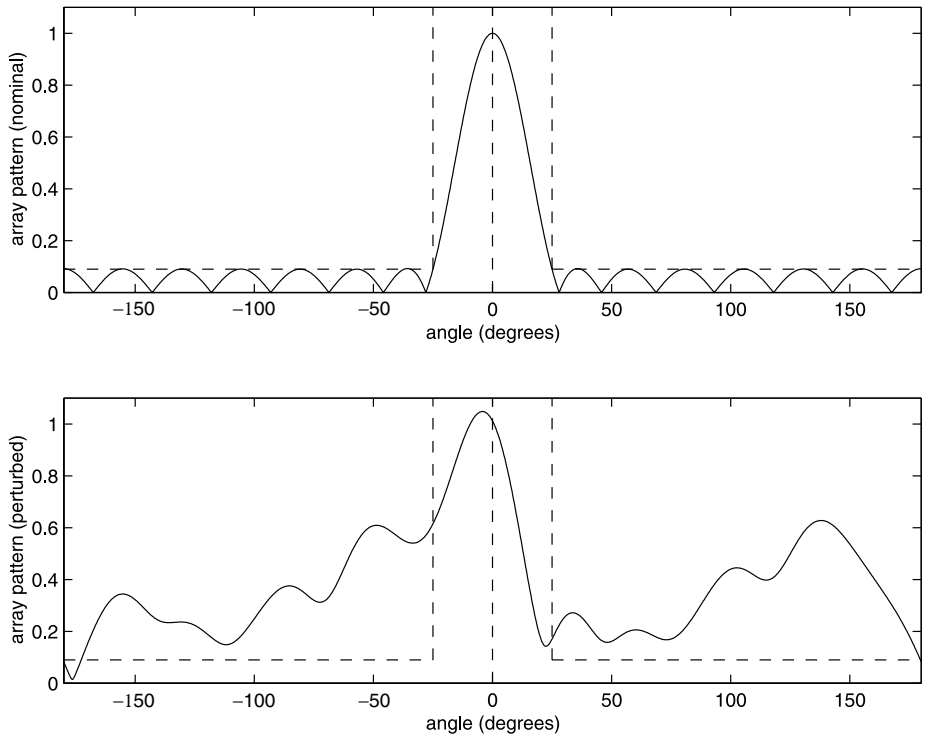


Figure 56.15. Nonrobust conventional array design. (a) array response (ideal circular placements); (b) array response (perturbed placements).

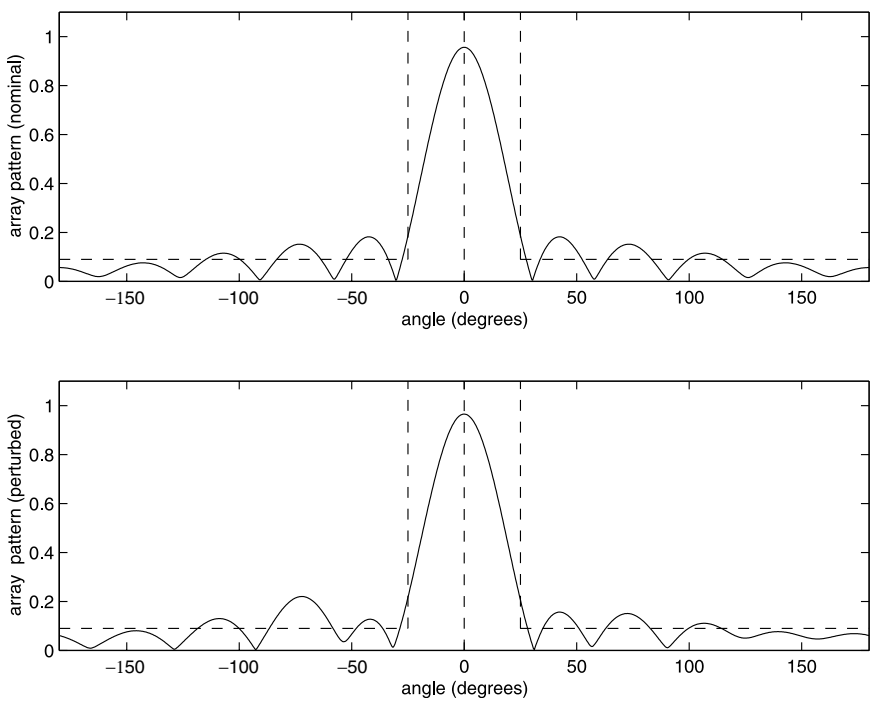


Figure 56.16. Robust SCOP array design: (a) array response (ideal circular placements); (b) array response (perturbed placements).

© 2005 by Chapman & Hall/CRC

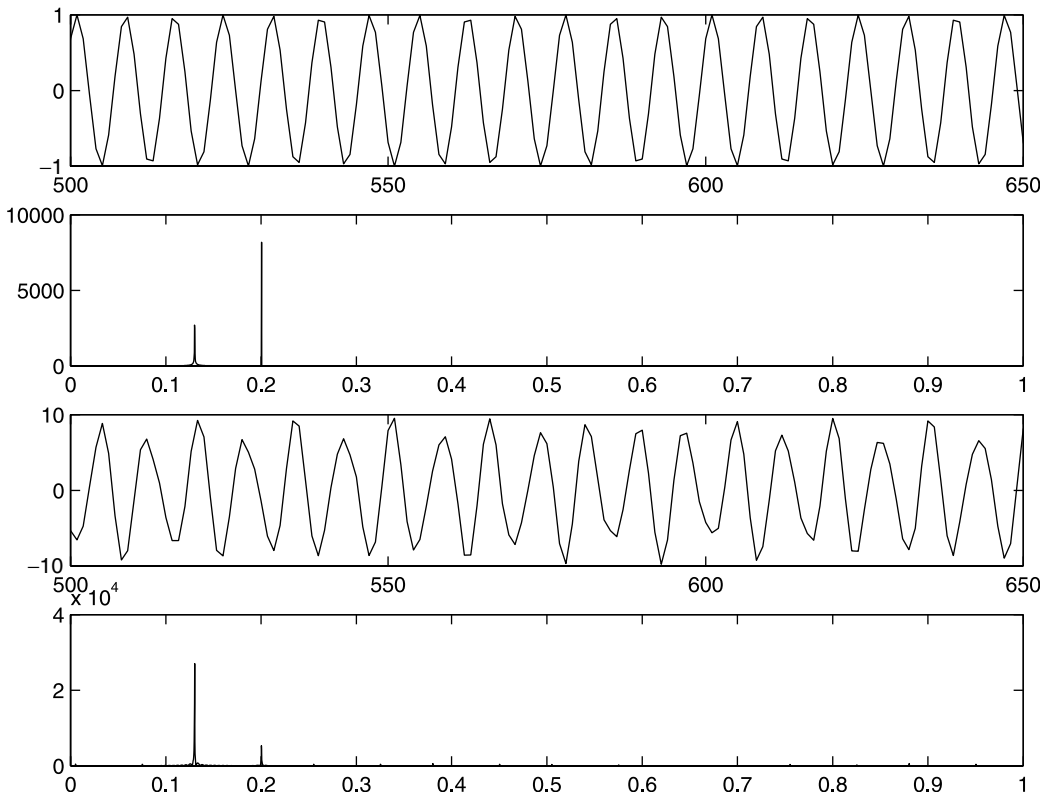


Figure 56.17. Wideband beamformer with numbers of channels, FFT size, decimation and interpolation factors set to eight: (a) section of desired signal in time domain; (b) magnitude of beamformer input; (c) section of beamformer output in time domain; (d) magnitude of beamformer output.

wideband signals into sets of narrowband signals. The signals in each subbband can then be decimated and processed at lower sampling rate using an analysis filter and reconstructed in a synthesis filer in conjunction with efficient polyphase filtering [47]. As an example, consider two tone signals of length 4096 at the input of a wideband beamfomer. The desired signal has unit amplitude and a frequency of $f_1 = 1300\,\text{Hz}$ at a DOA angle of 30°, while the interference has an amplitude of 2 with a frequency of $f_2 = 2002\,\text{Hz}$ at a DOA angle of 50°. The sampling rate is take at $f_s = 10{,}000\,\text{Hz}$. The number of channels, FFT size, the decimation, and interpolation factors are all taken to be eight and a polyphase finite impulse response filter of length 120 is used. Figure 56.17(a) shows a section of the desired signal in the time domain, Figure 56.17(b) shows the magnitude of the two signals at the input to the wideband beamformer, Figure 56.17(c) shows a section of the time-domain beamformer output with a slightly distorted desired signal, and Figure 56.17(d) shows the magnitude of the beamformer output with the interference signal greatly suppressed with respect to the desired signal. In a practical system [44,45] the array weights can be implemented using a field programmable gate array architecture and various adaptive schemes need to be used to update the weights.

56.4.2 iPAQS Implementation of an Acoustic Wideband Beamformer

In this section we consider the implementation of a Linux-based wireless networked acoustic sensor array testbed, utilizing commercially available iPAQs with built-in microphones, codecs, and

© 2005 by Chapman & Hall/CRC

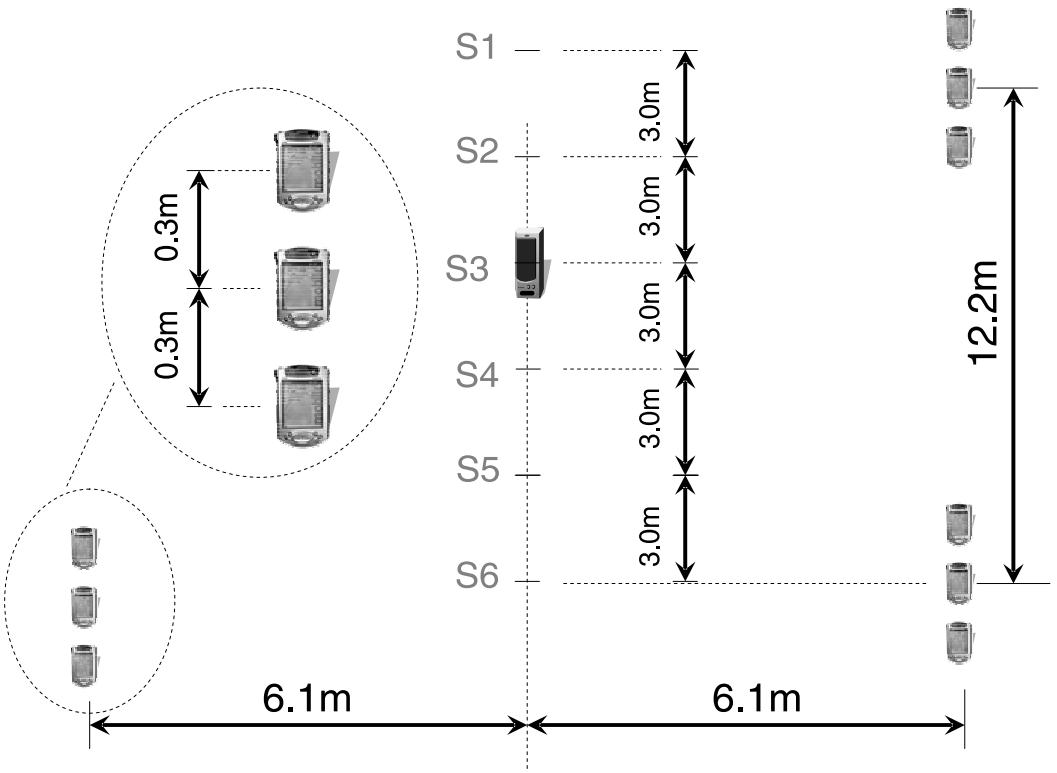


Figure 56.18. Linear subarray configuration.

microprocessors, plus wireless Ethernet cards, to perform acoustic source localization. Owing to space limitation, we only consider the cases where several subarrays are available to obtain independent DOA estimation of the same source(s), and the bearing crossings from the subarrays are used to obtain the location estimate. Direct localization for sources in the near field is also possible and has been demonstrated successfully [14]. We consider the AML and the time-delay-based CLS DOA estimator [30], where the CLS method uses the LS solution with a constraint to improve the accuracy of estimation. In the case of multiple sources, only the AML method can perform estimation and the alternating projection procedure is applied.

The first experimental setting is depicted in Figure 56.18, where three linear subarrays, each with three iPAQs, form the sensor network. In this far-field case (relative to each subarray), the DOA of the source is independently estimated in each subarray and the bearing crossing is used to obtain the location estimate. The speaker is placed at four distinct source locations $S1, \ldots, S6$, simulating source movement, and the same vehicle sound is played each time. Figure 56.19 depicts one snapshot (for clear illustration) of the AML and CLS results at six distinct source locations. We note that better results are clearly obtained when the source is inside the convex hull of the overall array. The second experimental setting is depicted in Figure 56.20, where four square subarrays each with four iPAQs form a single network. Two speakers, one playing the vehicle sound and the other one playing the music sound simultaneously, are placed inside the convex hull of the overall array. When both sources are playing simultaneously, as shown in Figure 56.21, the AML method is able to estimate both source locations simultaneously with alternating projection. Note that, when the number of subarray element increases, the localization accuracy of the results reported above improves, which agrees with the CRB analysis reported by Stoica and Nehorai [48] and in the previous section.

© 2005 by Chapman & Hall/CRC

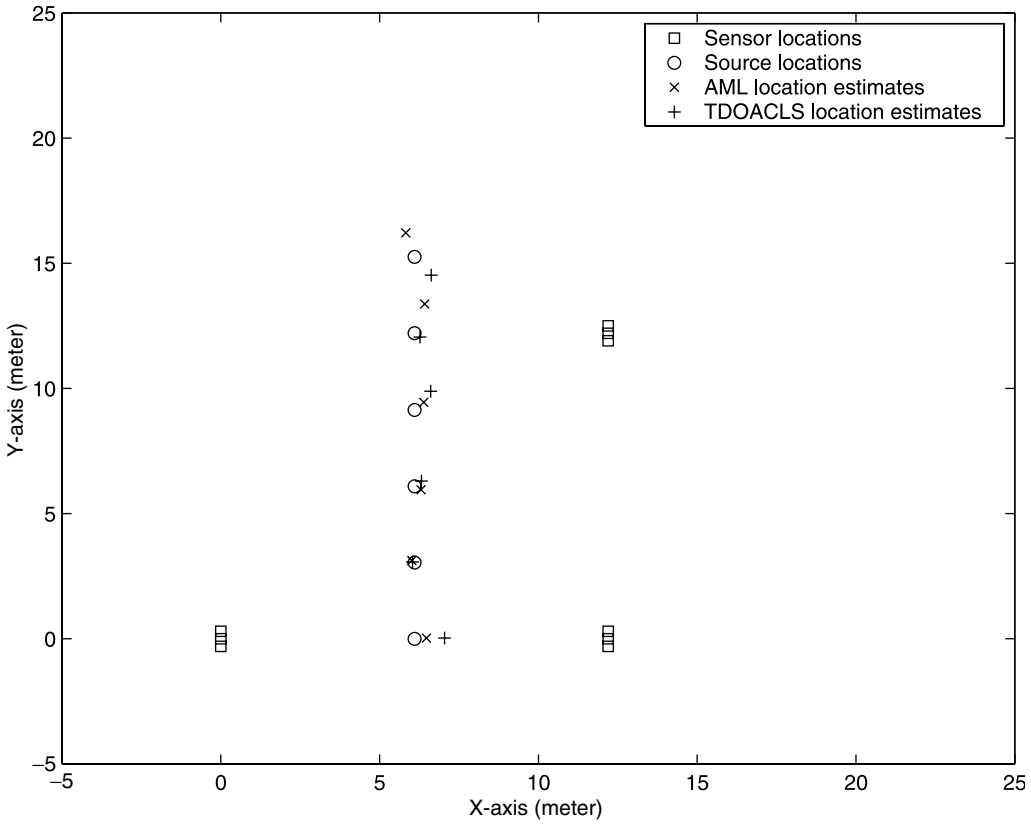


Figure 56.19. Cross-bearing localization of one source at different locations.

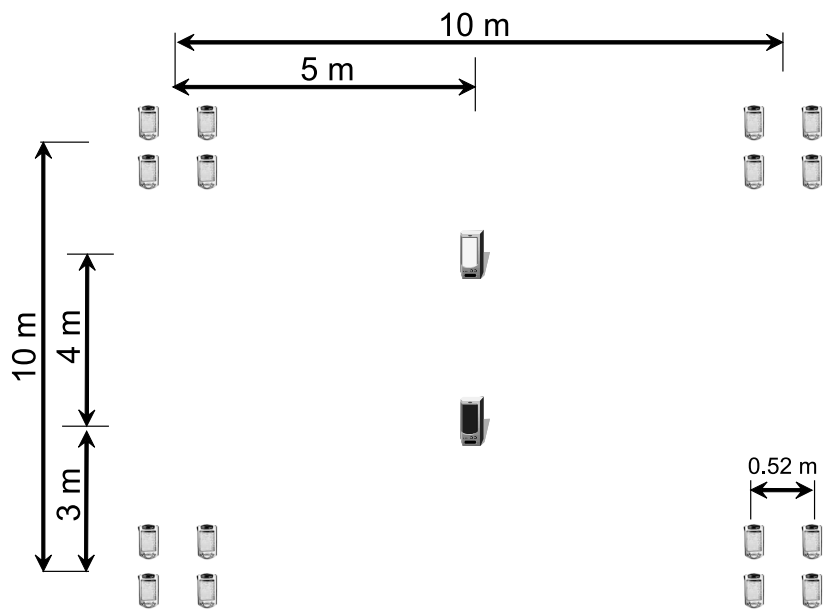


Figure 56.20. Square subarray configuration.

© 2005 by Chapman & Hall/CRC

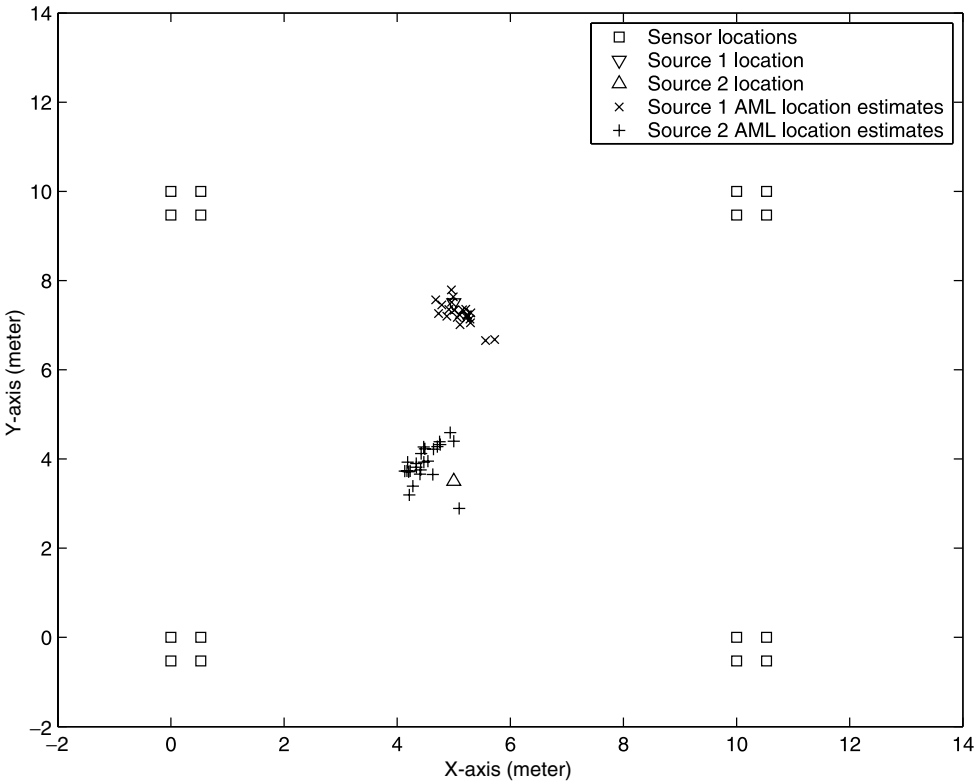


Figure 56.21. AML cross-bearing localization of two sources using alternating projection.

References

[1] Friis, H.T. and Feldman, C.B.A., A multiple unit steerable antenna for shear-wave reception, *Proceedings of the IRE*, 25, 841, 1937.

[2] Fourikis, N., *Advanced Array Systems, Applications, and RF Technologies*, Academic Press, 2000.

[3] Agre, J. and Clare, L., An integrated architecture for cooperative sensing and networks, *Computer*, 33, 106, 2000.

[4] Pottie, G.J. and Kaiser, W.J., Wireless integrated network sensors, *Communications of the ACM*, 43, 51, 2000.

[5] Kumar, S. *et al.* (eds), Special Issue on Collaborative Signal and Information Processing in Microsensor Networks, *IEEE Signal Processing Magazine*, 19, 13, 2002.

[6] Iyengar, S.S. and Kumar, S. (eds), Special Issue on Advances in Information Technology for High Performance and Computational Intensive Distributed Sensor Networks, *International Journal of High Performance Computing Applications*, 16, 203, 2002.

[7] Yao, K. *et al.* (eds), Special Issue on Sensor Networks, *European Journal on Applied Signal Processing*, 2003(4), 2003.

[8] Brandstein, M.S. and Silverman, H.F., A robust method for speech signal time-delay estimation in reverberant rooms, in *Proceedings of IEEE ICASSP*, vol. 1, 1997, 375.

[9] Yao, K. *et al.*, Blind beamforming on a randomly distributed sensor array system, *IEEE Journal on Selected Areas in Communication*, 16(8), 1555, 1998.

[10] Ward, D.B. and Williamson, R.C., Particle filter beamforming for acoustic source localization in a reverberant environment, in *Proceedings of IEEE ICASSP*, vol. 2, May 2002, 1777.

© 2005 by Chapman & Hall/CRC

[11] Aarabi, P., The fusion of distributed microphone arrays for sound localization, *European Journal on Applied Signal Processing*, 2003(4), 338, 2003.
[12] Zhao, F. *et al.*, Information-driven dynamic sensor collaboration, *IEEE Signal Processing Magazine*, 19, 61, 2002.
[13] Chen, J.C. *et al.*, Source localization and beamforming, *IEEE Signal Processing Magazine*, 19, 30, 2002.
[14] Chen, J.C. *et al.*, Coherent acoustic array processing and localization on wireless sensor networks, *Proceedings of the IEEE*, 91, 1154, 2003.
[15] Capon, J., High-resolution frequency–wavenumber spectrum analysis, *IEEE Proceedings*, 57, 1408, 1969.
[16] Schmidt, R.O., Multiple emitter location and signal parameter estimation, *IEEE Transactions on Antennas and Propagation*, AP-34(3), 276, 1986.
[17] Carter, G.C. (ed.), *Coherence and Time Delay Estimation: An Applied Tutorial for Research, Development, Test, and Evaluation Engineers*, IEEE Press, 1993.
[18] Emile, B. *et al.*, Estimation of time-delays with fewer sensors than sources, *IEEE Transactions on Signal Processing*, 46(7), 2012, 1998.
[19] Chen, J.C. *et al.*, Maximum-likelihood source localization and unknown sensor location estimation for wideband signals in the near-field, *IEEE Transactions on Signal Processing*, 50(8), 1843, 2002.
[20] Krim, J. and Viberg, M., Two decades of array signal processing research: the parametric approach, *IEEE Signal Processing Magazine*, 13, 67, 1996.
[21] Jaffer, A.G., Maximum likelihood direction finding of stochastic sources: a separable solution, in *Proceedings of IEEE ICASSP*, vol. 5, 1988, 2893.
[22] Chung, P.J. and Böhme, J.F., Comparative convergence analysis of EM and SAGE algorithms in DOA estimation, *IEEE Transactions on Signal Processing*, 49(12), 2940, 2001.
[23] Wang, H. and Kaveh, M., Coherent signal-subspace processing for the detection and estimation of angles of arrival of multiple wideband sources, *IEEE Transactions on Acoustics, Speech and Signal Processing*, ASSP-33, 823, 1985.
[24] Schau, H.C. and Robinson, A.Z., Passive source localization employing intersecting spherical surfaces from time-of-arrival differences, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-35(8), 1223, 1987.
[25] Smith, J.O. and Abel, J.S., Closed-form least-squares source location estimation from range-difference measurements, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-35(12), 1661, 1987.
[26] Chan, Y.T. and Ho, K.C., A simple and efficient estimator for hyperbolic location, *IEEE Transactions on Signal Processing*, 42(8), 1905, 1994.
[27] Brandstein, M.S. *et al.*, A closed-form location estimator for use with room environment microphone arrays, *IEEE Transactions on Speech and Audio Processing*, 5(1), 45, 1997.
[28] Chen, J.C. *et al.*, Source localization and tracking of a wideband source using a randomly distributed beamforming sensor array, *International Journal of High Performance Computing Applications*, 16(3), 259, 2002.
[29] Huang, Y. *et al.*, Passive acoustic source localization for video camera steering, in *Proceedings of IEEE ICASSP*, vol. 2, 2000, 909.
[30] Wang, H. *et al.*, A wireless time-synchronized COTS sensor platform part II: applications to beamforming, in *Proceedings of IEEE CAS Workshop on Wireless Communications and Networking*, September 2002.
[31] Brandstein, M.S. and Ward, D.B., *Microphone Arrays: Techniques and Applications*, Springer-Verlag, 2001.
[32] Tung, T.L. *et al.*, Source localization and spatial filtering using wideband MUSIC and maximum power beamforming for multimedia applications, in *Proceedings of IEEE SiPS*, October 1999, 625.

© 2005 by Chapman & Hall/CRC

[33] Kay, S.M., *Fundamentals of Statistical Signal Processing: Estimation Theory*, Prentice-Hall, New Jersey, 1993.
[34] Weiss, A.J. *et al.*, Analysis of signal estimation using uncalibrated arrays, in *Proceedings of IEEE ICASSP*, vol. 3, 1995, 1888.
[35] Cox, H. *et al.*, Robust adaptive beamforming, *IEEE Transactions Signal Processing*, 35, 1335, 1987.
[36] Er, M.H. and Cantoni, T., An alternative formulation for an optimum beamformer with robust capacity, *Proceedings of IEE Radar, Sonar, and Navagition*, 32, 447, 1985.
[37] Lebret, S. and Boyd, S., Antenna array pattern synthesis via convex optimization, *IEEE Transactions on Signal Processing*, 45, 5256, 1997.
[38] Wang, F. *et al.*, Optimal array synthesis using semidefinite programming, *IEEE Transactions on Signal Processing*, 51, 1172, 2003.
[39] Vorobyov, S.A. *et al.*, Robust adaptive beamforming using worst-case performance optimization: a solution to the signal mismatch problem, *IEEE Transactions on Signal Processing*, 51, 313, 2003.
[40] Vandenberghe, L. and Boyd, S., Semidefinite programming, *SIAM Review*, 38, 49095, 1996.
[41] Boyd, S. and Vandenberghe, L., Convex optimization, Class lecture notes, 2003.
[42] Sturm, J.F., CeDuMi: a Matlab toolbox for optimization over symmetric cones, http:fewcal.kub.nl/sturm/software/sedumi.html, 2001 (last accessed in February 2004).
[43] Vandenberghe, L., personal communication, September 2003.
[44] Moeller, T.J., Field programmable gate arrays for radar front-end digital signal processing, M.S. Dissertation, Massachusetts Institute of Technology, Cambridge, MA, 1999.
[45] Martinez, D.R. *et al.*, Application of reconfigurable computing to a high performance front-end radar signal processor, *Journal of VLSI Signal Processing*, 28(1–2), 65, 2001.
[46] Rabankin, D.V. and Pulsone, N.B., Subband-domain signal processing for radar array systems, *Proceedings of SPIE*, 3807, 174, 1999.
[47] Mitra, S.K., *Digital Signal Processing: A Computer-Based Approach*, 2nd ed., McGraw-Hill, 2001.
[48] Stoica, P. and Nehorai, A., MUSIC, maximum likelihood, and Cramer–Rao bound, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37, 720, 1989.

© 2005 by Chapman & Hall/CRC