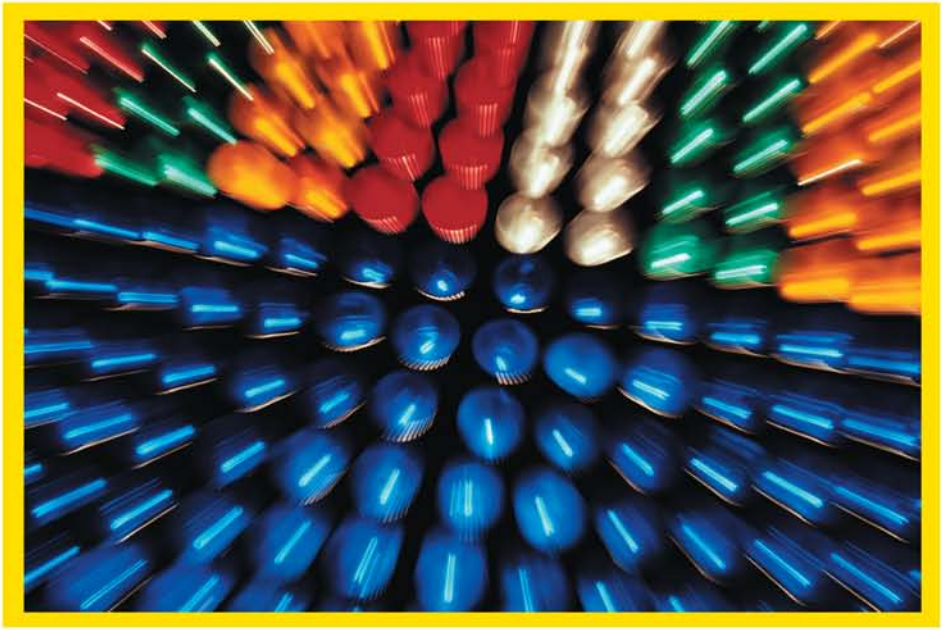
Auerbach Publications
Taylor & Francis Group

Handbook on Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless, and Peer-to-Peer Networks



Edited by Jie Wu

Handbook on Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless, and Peer-to-Peer Networks

Handbook on Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless, and Peer-to-Peer Networks

Edited by Jie Wu

Auerbach Publications
Taylor & Francis Group
Boca Raton London New York Singapore

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

Version Date: 20131031

International Standard Book Number-13: 978-0-203-32368-7 (eBook - PDF)

Preface

Overview

Theoretical and algorithmic approaches in Sensor networks, Ad hoc wireless networks, and Peer-to-peer networks (together called SAP networks) have played a central role in the development of emerging network paradigms. These three networks are characterized by their ad hoc nature without infrastructure or centralized administration. Unlike infrastructured networks, such as cellular networks, where nodes interact through a centralized base station, nodes in a SAP network interact in a peer-to-peer fashion. As a result of the mobility (including joining/leaving the network) of their nodes, SAP networks are characterized by dynamically changing topologies. The applications of SAP networks range from civilian (file-sharing) to disaster recovery (search-and-rescue), to military (battlefield).

The main goal of this book is to fill the need for comprehensive reference material on the recent development on theoretical and algorithmic aspects of three related fields. Topics covered include: theoretical and algorithmic methods/tools for optimization, computational geometry, graph theory, and combinatorics; protocol security and privacy; scalability design; distributed and localized solutions; database and data management; operating systems and middleware support; power control systems and energy efficient design; applications; and performance and simulations.

This book brings together different research disciplines to initiate a comprehensive technical discussion on theoretical and algorithmic approaches to three related fields: sensor networks, ad hoc wireless networks, and peer-to-peer networks. The objective is to identify several common theoretical and algorithmic approaches that can address issues related to SAP networks. The central topic is defined by the following two questions: What are the central technical issues in SAP networks? What are the possible solutions/tools available to address these issues?

This book is expected to serve as a reference book for developers in the telecommunication industry or as a textbook for a graduate course in computer science and engineering. It is organized in the following three groups as 47 chapters.

- Ad-Hoc Wireless Networks (19 chapters)
- Sensor Networks (16 chapters)
- Peer-to-Peer Networks (12 chapters)

Although many books have emerged recently in this area, none of them address all three fields in terms of common issues. This book has the following features and benefits:

- Coverage of three related fields, ad hoc wireless, sensor, and peer-to-peer networks, allows the reader to easily cross-reference similar results in three fields.
- International groups of authors present balanced coverage of research results.
- Systematic treatment of theoretical and algorithmic aspects allows the reader easy access to some important results.

- Applications and uses of these networks offer good motivation for research in these fields.
- Authoritative materials on a broad range of topics provide a comprehensive treatment of various important topics by some of the leading researchers in the field.

Common Theoretical and Algorithmic Issues

The following preliminary set of common theoretical and algorithmic issues is identified for SAP networks.

Location Management (in sensor networks, ad hoc wireless networks, and peer-to-peer networks): This issue addresses the problem of "Where is X." This problem can be analyzed from two aspects: *update* and *page*. The updating process notifies the location servers of the current locations of nodes. In search of a node, the paging process queries the servers to identify the exact/possible locations of the mobile station before the actual search. This avoids the potentially high costs of doing a global search. Updating and paging costs are tradeoffs. More frequent updates can improve the accuracy of the information in location servers, thus reducing the paging costs. On the contrary, less frequent updates can save updating costs, but may incur higher paging costs, especially for highly mobile stations. Many analytical tools such as queueing analysis and Markov chain analysis are used in this area. Graph theoretical models are used in peer-to-peer networks based on building an overlay network.

Security and Privacy (in sensor networks, ad hoc wireless networks, and peer-to-peer networks): Security is the possibility of a system withstanding an attack. There are two types of security mechanisms: preventive and detective. The majority of the preventive mechanisms have cryptography as building components. The goal of system security is to have controlled access to resources. The key requirements for SAP networks are confidentiality, authentication, integrity, non-repudiation and availability. SAP networks are more prone to attack because of their dynamic and/or infrastructureless nature. The attacks on networks can be categorized into interruption, interception, modification, and fabrication. In addition to various "attacks," a number of "trust" issues also occur in SAP networks. Cryptographic algorithms are widely used in this area.

Topology Design and Control (in sensor networks, ad hoc wireless networks, and peer-to-peer networks): Topology design deals with the way to control the network topology to achieve several desirable properties in SAP networks, including small diameter and small average node distance in peer-to-peer networks, and a certain level of node connectivity in sensor and ad hoc wireless networks. In general, each node has a similar number of neighbors, and the average nodal degree should be small. Regular and uniform structures are usually preferred. In many cases, topology control is tied to energy-efficient design. Traditional graph theory is usually used to deal with topology control.

Scalable Design (in sensor networks, ad hoc wireless networks, and peer-to-peer networks): Scalable design deals with how to increase the number of nodes without degrading system or protocol performance. The most common approach for supporting scalability is the clustering approach used both in sensor networks and ad hoc wireless networks. Basically, the network is partitioned into a set of clusterheads, with one clusterhead in each cluster. Clusterheads do not have direct connectivity to each other, but each clusterhead directly connects to all of its members. In sensor networks, the clustering approach is used to reduce the number of forward nodes (which contact the base station directly), and hence, to reduce overall energy consumption. The traditional scalability analysis is normally used.

Energy-Aware Design (in sensor networks and ad hoc wireless networks): Energy-aware design has been applied to various levels of protocol stacks. Most works have been done at the network layer. Several different protocols have been proposed to manage energy consumption by adjusting transmission ranges. In the source-independent approach, all nodes can be a source and are able to reach all other nodes by assigning appropriate ranges. The problem of minimizing the total transmission power consumption

(based on an assigned model) is NP-complete for both 2-D and 3-D space. Various heuristic solutions exists for this problem. At the MAC layer, power saving techniques for ad hoc and sensor networks can be divided into two categories: *sleeping* and *power controlling*. The sleeping methods put wireless nodes into periodic sleep states in order to reduce power consumption in the *idle listening* mode. Both graph theory and optimization methods are widely used in this area.

Routing and Broadcasting (in sensor networks and ad hoc wireless networks): This issue deals with trade-offs between proactive and reactive routing, flat and hierarchical routing, location-assist and non-location-assist routing and source-dependent and source-independent broadcasting. These trade-offs focus on cost and efficiency and are dependent on various parameters, such as network topology, host mobility, and network and traffic density. Various graph theoretical models (such as dominating set) and computational geometric models (such as Yao graph, RNG (relative neighborhood graph), and Gabriel graph) have been used. Graph theory, distributed algorithms, and computational geometry are widely used in this area.

Acknowledgments

I wish to thank all the authors for their contributions to the quality of the book. The support from NSF for an international workshop, held at Fort Lauderdale, Florida in early 2004, is greatly appreciated. Many chapters come from the extension of presentations at that workshop.

Special thanks to Rich O'Hanley, the managing editor, for his guidance and support throughout the process. It has been a pleasure to work with Andrea Demby and Claire Miller, who collected and edited all chapters. I am grateful to them for their continuous support and professionalism. Thanks to my students, Eyra Bethancourt and Max Haider, for their assistance.

Finally, I thank my children, NiNi and YaoYao, and my wife, Ruiguang Zhang, for making this all worthwhile and for their patience during my numerous hours working both at home and at the office.

Contributors

Mehran Abolhasan
Telecommunication and IT Research Institute (TITR)
University of Wollongong
Wollongong, NSW, Australia

Dharma P. Agrawal
OBR Research Center for Distributed and Mobile Computing
ECECS Department
University of Cincinnati
Cincinnati, Ohio

Anish Arora
Department of Computer Science and Engineering
The Ohio State University
Columbus, Ohio

James Aspnes
Department of Computer Science
Yale University
New Haven, Connecticut

Rimon Barr
Computer Science and Electrical Engineering
Cornell University
Ithaca, New York

Ratnabali Biswas
OBR Research Center for Distributed and Mobile Computing
ECECS Department
University of Cincinnati
Cincinnati, Ohio

Douglas M. Blough
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, Georgia

Andrija M. Bosnjakovic
Faculty of Electrical Engineering
University of Belgrade
Belgrade, Yugoslavia

Virgil Bourassa
Panthesis, Inc.
Bellevue, Washington

Aharon S. Brodie
Wayne State University
Detroit, Michigan

Gruia Calinescu
Department of Computer Science
Illinois Institute of Technology
Chicago, Illinois

Edgar H. Callaway, Jr.
Florida Communication Research Laboratory
Motorola Labs
Plantation, Florida

Guohong Cao
Department of Computer Science and Engineering
Pennsylvania State University
University Park, Pennsylvania

Ionu Cârdei
Department of Computer Science and Engineering
Florida Atlantic University
Boca Raton, Florida

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

Chih-Yung Chang
Department of Computer Science and Information Engineering
Tamkang University
Taipei, Taiwan

Sriram Chellappan
Department of Computer Science and Engineering
Ohio State University
Columbus, Ohio

Po-Yu Chen
Institute of Communications Engineering
National Tsing Hua University
Hsin-Chu, Taiwan

Wen-Tsuen Chen
Department of Computer Science
National Tsing Hua University
Hsin-Chu, Taiwan

Xiao Chen
Department of Computer Science
Texas State University
San Marcas, Texas

Yuh-Shyan Chen
Department of Computer Science and Information Engineering
National Chung Cheng University
Chia-Yi, Taiwan

Liang Cheng
Laboratory of Networking Group (LONGLAB)
Department of Computer Science and Engineering
Lehigh University
Bethlehem, Pennsylvania

Young-ri Choi
Department of Computer Sciences
The University of Texas at Austin
Austin, Texas

Marco Conti
Institute for Informatics and Telematics (IIT)
National Research Council (CNR)
Pisa, Italy

Jon Crowcroft
Computer Laboratory
University of Cambridge
Cambridge, UK

Arindam Kumar Das
Department of Electrical Engineering
University of Washington
Seattle, Washington

Saumitra M. Das
School of Electrical and Computer Engineering
Purdue University
West Lafayette, Indiana

Haitao Dong
Department of Computer Science and Technology
Tsinghua University
Beijing, China

Sameh El-Ansary
Swedish Institute of Computer Science (SICS)
Sweden

Mohamed Eltoweissy
Department of Computer Science
Virginia Tech
Falls Church, Virginia

Jakob Eriksson
Department of Computer Science and Engineering
University of California, Riverside
Riverside, California

Patrick Th. Eugster
Sun Microsystems, Inc.
Client Solutions Volketscuil Switzerland and School of Information and Communication Sciences
Swiss Federal Institute of Technology
Lausanne, Switzerland

Michalis Faloutsos
Department of Computer Science and Engineering
University of California, Riverside
Riverside, California

Yuguang Fang
Department of Electrical and Computer Engineering
University of Florida
Gainesville, Florida

Ophir Frieder
Department of Computer Science
Illinois Institute of Technology
Chicago, Illinois

Luca M. Gambardella
Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA)
Manno-Lugano
Switzerland

Mohamed G. Gouda
Department of Computer Sciences
The University of Texas at Austin
Austin, Texas

Aditya Gupta
OBR Research Center for Distributed and Mobile Computing
ECECS Department
University of Cincinnati
Cincinnati, Ohio

Sandeep K.S. Gupta
Department of Computer Science and Engineering
Arizona State University
Tempe, Arizona

Zygmunt J. Haas
Department of Electrical and Computer Engineering
Cornell University
Ithaca, New York

Joseph Y. Halpern
Department of Computer Science
Cornell University
Ithaca, New York

Seif Haridi
Royal Institute of Technology (IMIT/KTH)
Sweden

Fred B. Holt
Panthesis, Inc.
Bellevue, Washington

Jennifer C. Hou
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois

Hung-Chang Hsiao
Computer and Communications Research Center
National Tsing-Hua University
Hsin-Chu, Taiwan

Jinfeng Hu
Department of Computer Science and Technology
Tsinghua University
Beijing, China

Y. Charlie Hu
School of Electrical and Computer Engineering
Purdue University
West Lafayette, Indiana

Yiming Hu
Department of Electrical and Computer Engineering and Computer Science
University of Cincinnati
Cincinnati, Ohio

Chi-Fu Huang
Department of Computer Science and Information Engineering
National Chiao Tung University
Hsin-Chu, Taiwan

Zhuochuan Huang
Department of Computer and Information Sciences
University of Delaware
Newark, Delaware

François Ingelrest
IRCICA/LIFL
University of Lille
INRIA futurs
France

Neha Jain
OBR Research Center for Distributed and Mobile Computing
ECECS Department
University of Cincinnati
Cincinnati, Ohio

Xiaohua Jia
Department of Computer Science
City University of Hong Kong
Hong Kong

Kennie Jones
Department of Computer Science
Old Dominion University
Norfolk, Virginia

Dongsoo S. Kim
Electrical and Computer Engineering
Indiana University, Purdue University
Indianapolis, Indiana

Chung-Ta King
Department of Computer Science
National Tsing-Hua University
Hsin-Chu, Taiwan

Manish Kochhal
Department of Electrical and Computer Engineering
Wayne State University
Detroit, Michigan

Odysseas Koufopavlou
Electrical and Computer Engineering Department
University of Patras
Greece

Srikanth Krishnamurthy
Department of Computer Science and Engineering
University of California, Riverside
Riverside, California

Tom La Porta
Pennsylvania State University
University Park, Pennsylvania

Mauro Leoncini
Dipartimento dilngegneria dell' in Formazione
UniversitÂ di Modena e Reggio Emilia
Modena, Italy

Dongsheng Li
School of Computer
National University
of Defense Technology
Changsha, China

Li (Erran) Li
Center for Networking
Research
Bell Labs, Lucent
Holmdel, New Jersey

Xiang-Yang Li
Department of Computer
Science
Illinois Institute of Technology
Chicago, Illinois

Xiuqi Li
Department of
Computer Science
and Engineering
Florida Atlantic University
Boca Raton, Florida

Hai Liu
Department of Computer
Science
City University of
Hong Kong
Hong Kong

Xuezheng Liu
Department of Computer
Science and Technology
Tsinghua University
Beijing, China

Yunhao Liu
Department of
Computer Science
and Engineering
Michigan State
University
East Lansing, Michigan

Xicheng Lu
School of Computer
National University of
Defense Technology
Changsha, China

B. S. Manoj
Department of
Computer Science
and Engineering
Indian Institute of
Technology
Chennai, India

Gaia Maselli
Institute for Informatics and
Telematic (IIT) National
Research Council (CNR)
Pisa, Italy

Jelena Mišić
University of Manitoba
Winnipeg, Manitoba
Canada

Vojislav B. Mišić
University of Manitoba
Winnipeg, Manitoba
Canada

Nikolay A. Moldovyan
Specialized Center of
Program Systems
(SPECTR)
St. Petersburg, Russia

Roberto Montemanni
Istituto Dalle Molle di Studi
sull'Intelligenza Artificiale
(IDSIA)
Manno-Lugano, Switzerland

Thomas Moscibroda
Department of Computer
Science
Swiss Federal Institute of
Technology
Zurich, Switzerland

Anindo Mukherjee
OBR Research Center for
Distributed and Mobile
Computing
ECECS Department
University of Cincinnati
Cincinnati, Ohio

C. Siva Ram Murthy
Department of
Computer Science
and Engineering
Indian Institute of
Technology
Chennai, India

Lionel M. Ni
Department of Computer
Science
Hong Kong University of
Science and Technology
Hong Kong

Stephan Olariu
Department of Computer
Science
Old Dominion University
Norfolk, Virginia

Shashi Phoha
Pennsylvania State
University
University Park,
Pennsylvania

Jovan Popovic
Faculty of Electrical
Engineering
University of Belgrade
Belgrade, Yugoslavia

Himabindu Pucha
School of Electrical and
Computer Engineering
Purdue University
West Lafayette, Indiana

Cauligi S. Raghavendra
Departments of Electrical Engineering-Systems and Computer Science
University of Southern California
Los Angeles, California

Giovanni Resta
Instituto di Informaticae Telematica
Area della Ricerca del CNR
Pisa, Italy

Paolo Santi
Instituto di Informaticae Telematica
Area della Ricerca del CNR
Pisa, Italy

Loren Schwiebert
Department of Computer Science
Wayne State University
Detroit, Michigan

Sandhya Sekhar
OBR Research Center for Distributed and Mobile Computing
ECECS Department
University of Cincinnati
Cincinnati, Ohio

Gauri Shah
IBM Almaden Research Center
San Jose, California

Chien-Chung Shen
Department of Computer and Information Sciences
University of Delaware
Newark, Delaware

Haiying Shen
Department of Electrical and Computer Engineering
Wayne State University
Detroit, Michigan

Jian Shen
Department of Mathematics
Texas State University
San Marcas, Texas

Jang-Ping Sheu
Department of Computer Science and Information Engineering
National Central University
Chung-Li, Taiwan

Shuming Shi
Department of Computer Science and Technology
Tsinghua University
Beijing, China

Weisong Shi
Department of Computer Science
Wayne State University
Detroit, Michigan

Zhenghan Shi
Department of Computer Science
Clemson University
Clemson, South Carolina

David Simplot-Ryl
IRCICA/LIFL
University of Lille
INRIA futurs
France

Nicolas Sklavos
VLSI Design Laboratory
University of Patras
Patras, Greece

Pradip K Srimani
Department of Computer Science
Clemson University
Clemson, South Carolina

Ivan Stojmenovic
Computer Science
SITE
University of Ottawa
Ottawa, Ontario
Canada

Caimu Tang
Department of Computer Science
University of Southern California
Los Angeles, California

Yu-Chee Tseng
Department of Computer Science and Information Engineering
National Chiao Tung University
Hsin-Chu, Taiwan

Giovanni Turi
Institute for Informatics and Telematics (IIT) National Research Council (CNR)
Pisa, Italy

Robbert van Renesse
Department of Computer Science
Cornell University
Ithaca, New York

Ashraf Wadaa
Department of Computer Science
Old Dominion University
Norfolk, Virginia

Peng-Jun Wan
Department of Computer Science
Illinois Institute of Technology
Chicago, Illinois

Wheizhao Wang
Department of Computer Science
Illinois Institute of Technology
Chicago, Illinois

Guiling Wang
Department of Computer Science and Engineering
Pennsylvania State University
University Park, Pennsylvania

Xun Wang
Department of Computer Science and Engineering
Ohio State University
Columbus, Ohio

Roger Wattenhofer
Department of Computer Science
Swiss Federal Institute of Technology
Zurich, Switzerland

Larry Wilson
Department of Computer Science
Old Dominion University
Norfolk, Virginia

Jie Wu
Department of Computer Science and Engineering
Florida Atlantic University
Boca Raton, Florida

Tadeusz Wysocki
Telecommunication and IT Research Institute (TITR)
University of Wollongong
Wollongong, New South Wales
Australia

Li Xiao
Department of Computer Science and Engineering
Michigan State University
East Lansing, Michigan

Cheng-Zhong Xu
Department of Electrical and Computer Engineering
Wayne State University
Detroit, Michigan

Chuanfu Xu
School of Computer
National University of Defense Technology
Changsha, China

Dong Xuan
Department of Computer Science and Engineering
Ohio State University
Columbus, Ohio

Qing Ye
Laboratory of Networking Group (LONGLAB)
Department of Computer Science and Engineering
Lehigh University
Bethlehem, Pennsylvania

Hongqiang Zhai
Department of Electrical and Computer Engineering
University of Florida
Gainesville, Florida

Honghai Zhang
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois

Wensheng Zhang
Pennsylvania State University
University Park, Pennsylvania

Weimin Zheng
Department of Computer Science and Technology
Tsinghua University
Beijing, China

Yingwu Zhu
Department of Electrical and Computer Engineering and Computer Science
University of Cincinnati
Cincinnati, Ohio

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

Contents

I

Ad Hoc Wireless Networks

The maturity of wireless transmissions and the popularity of portable computing devices have made the dream of "communication anytime and anywhere" possible. An ad hoc wireless network is a good choice for fulfilling this dream. An ad hoc wireless network consists of a set of mobile hosts operating without the aid of an established infrastructure of centralized administration. Communication is done through wireless links among mobile hosts using their antennas. Due to concerns such as radio power limits and channel utilization, a mobile host may not be able to communicate directly with other hosts in a single-hop fashion. In this case, a multihop scenario occurs, in which the packets sent by the source host must be relayed by several intermediate hosts before reaching the destination host.

Although military tactical communication is still considered the primary application for ad hoc wireless networks, commercial interest in this type of network continues to grow. Applications such as law enforcement operation, commercial and educational use, and sensor networks are just a few possible commercial examples. There are several technical challenges related to the ad hoc wireless network. In ad hoc wireless networks, the topology is highly dynamic, and frequent changes in the topology may be difficult to predict. With the use of wireless links, the network suffers from higher loss rates, and can experience more delays and jitter. In addition, physical security is limited due to wireless transmission. Finally, as ad hoc wireless network nodes rely on batteries, energy saving is an important system design criterion.

Most of the existing works on ad hoc wireless networks focus on issues related to the network layer, such as routing and broadcasting. Routing protocols in ad hoc wireless networks are either *proactive* or *reactive*, although a combination of proactive and reactive is also possible. In proactive routing, routes to all destinations are computed *a priori* and are maintained in the background via a periodic update process. Route information is maintained either as routing tables or as global link state information. In reactive routing, a route to a specific destination is computed "on demand," that is, only when needed. To efficiently use resources in controlling large dynamic networks, hierarchical routing, including cluster based and dominating set based, is normally used.

Many other technical issues are discussed through the use of protocol stacks where at least four layers are used:

1. Physical layer: responsible for frequency selection, carrier frequency generation, signal detection, modulation, and data encryption.
2. Data link layer: responsible for the multiplexing of data streams, data frame detection, medium access, and error control.
3. Network layer: responsible for forwarding the data to appropriate destinations.
4. Application layer: responsible for supporting various applications.

The 19 chapters in this section cover a wide range of topics across multiple layers: MAC (part of the data link layer), network, and applications. One chapter is devoted to the cross-layer architecture for ad hoc wireless networks. Several chapters deal with various techniques for efficient and scalable routing, including multicasting and geocasting, in ad hoc wireless networks. One chapter discusses routing in a selfish wireless network. Three chapters present some recent results on topology control while three other chapters are dedicated to energy-efficient design under several different system settings. The security and reliability issues are covered in two separate chapters. MAC protocols are given in one dedicated chapter. Of the three chapters about applications, one discusses ad hoc relaying in cellular networks, one uses ad hoc wireless networks for robust data communication, and one is devoted to the application in Bluetooth. This section ends with a chapter on scalable simulation for ad hoc wireless networks.

1

A Modular Cross-Layer Architecture for Ad Hoc Networks

Marco Conti
Jon Crowcroft
Gaia Maselli
Giovanni Turi

The success of the cleanly layered Internet Architecture has promoted its adoption for wireless and mobile networks, including ad hoc networks. This has also fostered skepticism toward alternative approaches. However, a strict-layered design is not flexible enough to cope with the dynamics of mobile networks and can prevent many classes of performance optimizations. To what extent, then, must developers modify the pure layered approach by introducing closer cooperation among protocols belonging to different layers?

Although the debate on cross-layer versus legacy-layer architecture has been around for a while, we propose a novel solution based on loosely coupled cross-layering, which constitutes a trade-off between the two extremes. Our solution allows for performance optimizations, but at the same time maintains flexibility.

This innovative architecture not only makes room for techniques to design new ad hoc protocols, for which we present specific examples, but also opens up the possibility of research into the usage of cross-layering for the Internet more generally.

1.1 Introduction

The Internet transparently connects millions of heterogeneous devices, supporting a huge variety of communications. From a networking standpoint, its popularity is due to a core design that has made it extensible, and robust against evolving usage as well as failures. Now, mobile devices and wireless communications prompt the vision of networking without a network (ad hoc networking). This brings

new challenging issues where the need for flexibility confronts ad hoc constraints. A careful architectural design for the ad hoc protocol stack is necessary to incorporate this emerging technology.

The Internet architecture layers protocol and network responsibilities, breaking down the networking system into modular components, and allowing for transparent improvements of single modules. In a *strict-layered* system, protocols are independent of each other and interact through well-defined (and static) interfaces: each layer implementation depends on the interfaces available from the lower layer, and those exported to the upper layer. Strict-layering provides flexibility to a system's architecture: extensions introduced into single levels do not affect the rest of the system. The separation of concerns brings the added benefits of minimizing development costs by re-using existing code. This design approach relies on "horizontal" communication between peer protocol layers on the sender and receiver devices (the dashed arrows in Figure 1.1). The result is a trend to spend bandwidth (an abundant resource in the Internet) instead of processing power and storage.

Several aspects of the Internet architecture have led to the adoption of this strict-layer approach also for mobile ad hoc networks. Some of these aspects include (1) the "IP-centric" view of ad hoc networks; and (2) the flexibility offered by independent layers, which allows for reuse of existing software. The choice of the layered approach is supported by the fact that ad hoc networks are considered mobile extensions of the Internet, and hence the protocol stack must be suitable. However, this design principle clashes with the following facts:

1. Issues such as energy management, security, and cooperation characterize the whole stack and cannot be solved inside a single layer.
2. Ad hoc networks and the Internet have conflicting constraints; and while the former are dynamic, the latter is relatively static.

Some guidelines to approach these problems point to an enhancement of "vertical" communication in a protocol stack (see Figure 1.1),[1,2] as a way to reduce peer (horizontal) communication, and hence conserve bandwidth. Vertical communication, especially between nonadjacent layers, facilitates local data retrieval, otherwise carried out through network communication. The practice of accessing not only the next lower layer, but also other layers of the protocol stack, leads to *cross-layering* to allow performance improvements. The main downside of strict-layering is that it hinders extensibility: a new, higher-level component can only build on what is provided by the next lower layer.[3] Hence, if one layer needs to access functionality or information provided by a nonadjacent layer, then an intermediate extension should be devised. Cross-layering allows nonadjacent protocols to directly interact, making overall optimizations possible and achieving extensibility at the eventual expense of flexibility.

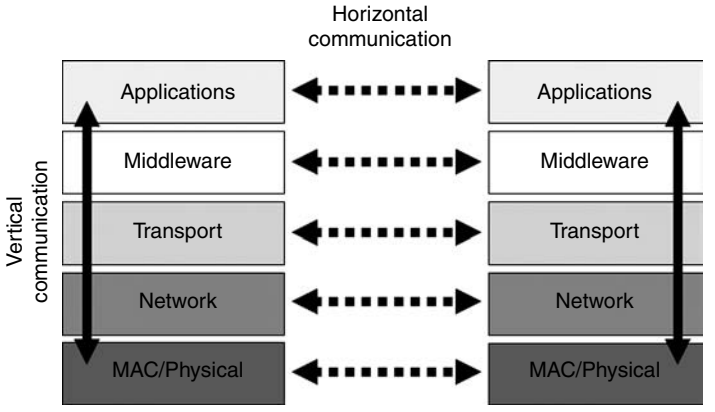


FIGURE 1.1 The Internet emphasizes horizontal communication between peer protocol layers to save router resources, while ad hoc networking promotes vertical interaction to conserve bandwidth.

In the literature there is much work showing the potential of cross-layering for isolated performance improvements in ad hoc networks. However, the focus of that work is on specific problems, as it looks at the joint design of two to three layers only. For example, cross-layer interactions between the routing and the middleware layers allow the two levels to share information with each other through system profiles, in order to achieve high quality in accessing data.[4] An analogous example is given by Schollmeier et al.,[5] where a direct interaction between the network and the middleware layers, termed Mobile Peer Control Protocol, is used to push a reactive routing to maintain existing routes to members of a peer-to-peer overlay network. Yuen et al.[6] propose an interaction between the MAC and routing layers, where information like signal-to-noise ratio, link capacity, and MAC packet delay is communicated to the routing protocol for the selection of optimal routes. Another example is the joint balancing of optimal congestion control at the transport layer with power control at the physical layer.[7] This work observes how congestion control is solved in the Internet at the transport layer, assuming that link capacities are fixed quantities. In ad hoc networks, this is not a good assumption, as transmission power, and hence throughput, can be dynamically adapted on each link. Last, but not least, Kozat et al.[8] propose cross-layer interaction between physical, MAC, and routing layers to perform joint power control and link scheduling as an optimized objective.

Although these solutions are clear examples of optimization introduced by cross-layering, the drawback on the resulting systems is that they contain tightly coupled, and therefore mutually dependent, components. Additionally, while an individual suggestion for cross-layer design, in isolation, may appear appealing, combining them all together could result in interference among the various optimizations.[9] From an architectural point of view, this approach leads to an "unbridled" stack design, difficult to maintain efficiently, because every modification must be propagated across all protocols. To give an example of interfering optimizations, let us consider an adaptation loop between a rate-adaptive MAC and minimal hop routing protocol (most ad hoc routing protocols are minimum hop). A rate-adaptive MAC would be able to analyze the quality of channels, suggesting higher layers on the outgoing links, which provide the higher data rates in correspondence with shorter distances. This conflicts with typical decisions of a minimum hop routing protocol, which chooses a longer link (for which the signal strength and data rate are typically lower) to reach the destination while using as few hops as possible.

We claim that cross-layering can be achieved, maintaining the layer separation principle, with the introduction of a vertical module, called *Network Status** (NeSt), which controls all cross-layer interactions (see Section 1.2). The NeSt aims at generalizing and abstracting vertical communications, getting rid of the tight coupling from an architectural standpoint. The key aspect is that protocols are still implemented in isolation inside each layer, offering the advantages of:

- Allowing for full compatibility with standards, as NeSt does not modify each layer's core functions
- Providing a robust upgrade environment, which allows the addition or removal of protocols belonging to different layers from the stack, without modifying operations at other layers
- Maintaining the benefits of a modular architecture (layer separation is achieved by standardizing access to the NeSt)

In addition to the advantages of a full cross-layer design, which still satisfies the layer separation principle, the NeSt provides full context awareness at all layers. Information regarding the network topology, energy level, local position, etc. is made available by the NeSt to all layers, to achieve optimizations, and offers performance gains from an overhead point of view. Although this awareness is restricted to the node's local view, protocols can be designed so as to adapt the system to highly variable network conditions (the typical ad hoc characteristic).

*This term indicates the collection of network information that a node gathers at all layers. It should not be confused with a concept of globally shared network context.

This innovative architecture opens research opportunities for techniques to design and evaluate new ad hoc protocols (see Section 1.3), but also remains compliant with the usage of legacy implementations, introducing new challenging issues concerning the usage of cross-layering for the Internet more generally (see Section 1.4).

1.2 Toward Loosely Coupled Cross-Layering

One of the main problems caused by direct cross-layer interactions (as already discussed in Section 1.1) is the resulting *tight coupling* of interested entities. To solve this problem, the NeSt stands vertically beside the network stack (as shown in Figure 1.2), handling eventual cross-layer interactions among protocols. That is, the NeSt plays the role of intermediary, providing standard models to design protocol interactions. While the new component uniformly manages vertical exchange of information between protocols, usual network functions still take place layer-by-layer through standardized interfaces, which remain unaltered. This introduced level of indirection maintains the *loosely coupled* characteristic of Internet protocols, preserving the flexible nature of a layered architecture.

The idea is to have the NeSt exporting an interface toward protocols, so as to allow sharing of information and reaction to particular events. In this way, cross-layer interactions do not directly take place between the interested protocols, but are implemented using the abstractions exported by the NeSt. This approach allows protocol designers to handle new cross-layer interactions apart, without modifying the interfaces between adjacent layers. The work described by Conti et al.[10,11] introduces this idea in the context of pure ad hoc networking. This work extends the definition of the NeSt interaction models, presenting the exported interface. This is to evolve toward a general-purpose component, eventually suitable for cross-layering in a future Internet architecture.

1.2.1 Overview of NeSt Functionalities

The NeSt supports cross-layering implementation with two models of interaction between protocols: *synchronous* and *asynchronous*. Protocols interact synchronously when they share private data (i.e., internal status collected during their normal functioning). A request for private data takes place on-demand, with a protocol querying the NeSt to retrieve data produced at other layers, and waiting for the result. Asynchronous interactions characterize the occurrence of specified conditions to which protocols may be willing to react. As such conditions are occasional (i.e., not deliberate), protocols are required to subscribe for their occurrences. In other words, protocols subscribe for events they are interested in, and then return to their work. The NeSt, in turn, is responsible for delivering eventual occurrences to the right subscribers. Specifically, we consider two types of events: *internal* and *external*. Internal events are directly generated inside the protocols. Picking just one example, the routing protocol notifies the rest of the stack about a "broken route" event, whenever it discovers the failure of a preexisting route. On the other side, external events are discovered inside the NeSt on the basis of instructions provided by subscriber protocols.

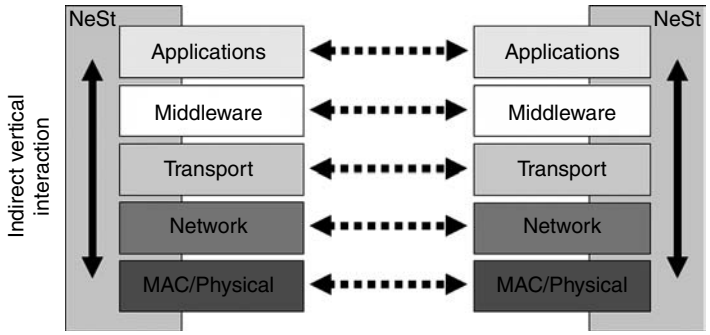


FIGURE 1.2 An architectural trade-off for loosely coupled vertical protocol interactions.

An example of an external event is a condition on the host energy level. A protocol can subscribe for a "battery-low" event, specifying an energy threshold to the NeSt, which in turn will notify the protocol when the battery power falls below the given value.

As the NeSt represents a level of indirection in the treatment of cross-layer interactions, an agreement for common-data and events representation inside the vertical component is a fundamental requirement. Protocols must agree on a common representation of shared information, in order to guarantee loose coupling. To this end, the NeSt works with *abstractions* of data and events, intended as a set of data structures that comprehensively reflect the relevant (from a cross-layering standpoint) information and special conditions used throughout the stack. A straightforward example is the topology information collected by a routing protocol. To abstract from implementation details of particular routing protocols, topology data can be represented as a graph inside the NeSt. Therefore, the NeSt becomes the provider of shared data, which appear independent of its origin and hence usable by each protocol.

How is protocol internal data exported into NeSt abstractions? The NeSt accomplishes this task using *callback* functions, which are defined and installed by protocols themselves. A callback is a procedure that is registered to a library (the NeSt interface) at one point in time, and later on invoked (by the NeSt). Each callback contains the instructions to encode private data into an associated NeSt abstraction. In this way, the protocol designer provides a tool for transparently accessing protocol internal data.

1.2.2 The NeSt Interface

To give a technical view of the vertical functionalities, we assume that the language used by the NeSt to interface the protocol stack allows for declaration of functions, procedures, and common data structures. We adopt the following notation to describe the NeSt interface:

$$functionName : (input) \rightarrow output$$

Each protocol starts its interaction with the NeSt by *registering* to the vertical component. This operation assigns to each protocol a unique identifier (PID), as shown by step *a* in Figure 1.3. The registration is expected to happen once for all at protocol bootstrap time by calling

$$register : () \rightarrow PID$$

As described in the previous section, the NeSt does not generate shared data, but acts as an intermediary between protocols. More precisely, a protocol *seizes* the NeSt abstractions related to its internal functionalities and data structures. The example of the network topology suggests the routing protocol to acquire ownership of an abstract graph containing the collected routing information. This operation requires a protocol to identify itself, providing the PID, and to specify the abstraction's identifier (AID) together with the associated callback function (see step *b* in Figure 1.3). When invoked, the callback function fills out the

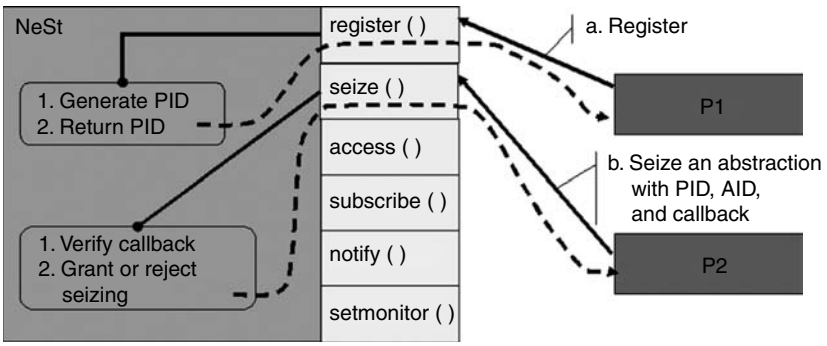


FIGURE 1.3 NeSt functionalities: register and seize.

abstraction, encoding protocol internal representation in NeSt format. Note that the callback invocation takes place asynchronously with the seizing operation, every time a fresh copy of the associated data is needed inside the NeSt. The entire process begins by calling

$$seize : (PID, AID, readCallBack()) \rightarrow result$$

The result of a call to $seize()$ indicates the outcome of the ownership request.

Once an abstraction has been seized, the NeSt is able to satisfy queries of interested entities. A protocol *accesses* an abstraction by calling

$$access : (PID, AID, filter()) \rightarrow result$$

This function shows that the caller must identify itself with a valid PID, providing also the abstraction identifier and a *filter* function. The latter parameter is a container of instructions for analyzing and selecting only information relevant to the caller's needs. The NeSt executes this call by spawning an internal computation that performs the following steps (see Figure 1.4):

1. Invoke the callback installed by the abstraction's owner (if any).
2. Filter the returned data locally (i.e., in the context of the NeSt).
3. Deliver the filtering result to the caller.

The remaining functions of the NeSt interface cope with asynchronous interactions. In the case of internal events, the role of the NeSt is to collect subscriptions, wait for notifications, and vertically dispatch occurrences to the appropriate subscribers, as shown in Figure 1.5. A protocol *subscribes* for an event by identifying itself and providing the event's identifier (EID), calling the function

$$subscribe : (PID, EID) \rightarrow result$$

To *notify* the occurrence of an event, a protocol must specify in addition to the event identifier (EID), information regarding the occurrence. This happens by calling the function

$$notify : (PID, EID, info) \rightarrow result$$

After the notification of an event, the NeSt checks it against subscriptions, and dispatches the occurrence to each subscriber.

In the case of external events, protocols subscribe by instructing the NeSt on how to detect the event. The rules to detect an external event are represented by a monitor function that periodically checks the status of a NeSt abstraction. When the monitor detects the specified condition, the NeSt dispatches the information to the subscriber protocol. As shown in Figure 1.6, a protocol delegates the *monitoring* of an

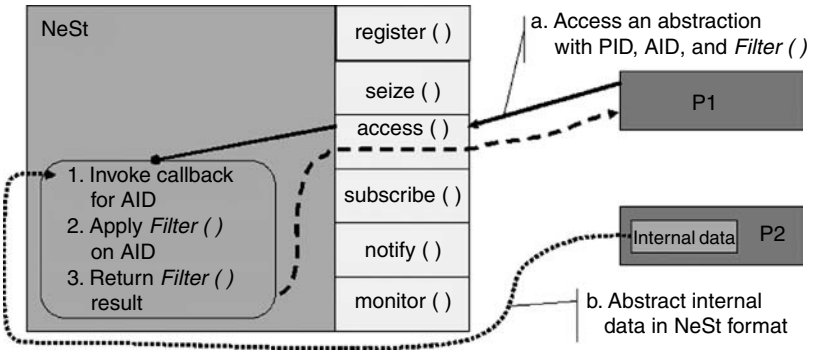


FIGURE 1.4 NeSt functionalities: access an abstraction.

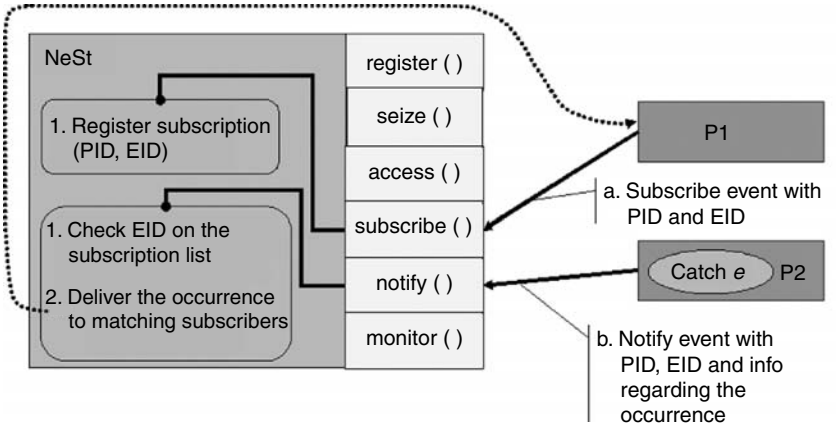


FIGURE 1.5 NeSt functionalities: management of internal events.

external event by passing to the NeSt a monitor function and the identifier of the target abstraction. This happens by calling

$$set\ monitor : (PID, AID, monitor()) \rightarrow result$$

The NeSt serves this call by spawning a *persistent* computation (see Figure 1.6) that executes the following steps:

1. Verify the monitor (e.g., type checking).
2. While (true):
   a. Refresh the abstraction invoking the associated callback.
   b. Apply the monitor to the resulting content.
   c. If the monitor detects the special condition, then notify the requesting protocol.

The result of a call to *set monitor*() only returns the outcome of the monitor's installation, while the notification of external events takes place asynchronously.

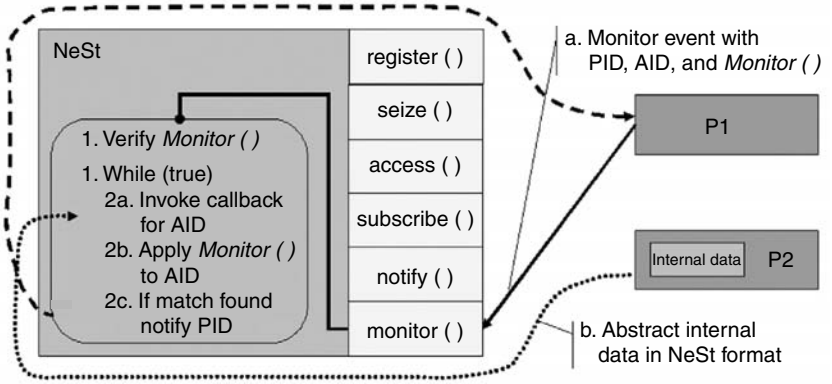


FIGURE 1.6 NeSt functionalities: management of external events.

1.2.3 Design and Implementation Remarks

It is difficult to find comparisons to the proposed architecture as, to the best of our knowledge, there are no similar approaches in the organization of a protocol stack. However, there are important observations and remarks to be given.

First of all, the NeSt is a component dedicated to enabling optimization. If on the one hand it helps maintain the layering principle allowing loosely coupled interactions, on the other hand it must guarantee the appropriate level of real-timing. That is, when subjected to a heavy load of cross-layering, the NeSt should be responsive, avoiding making protocol efforts fruitless. For example, in the case of synchronous interactions where call-backs are employed, the NeSt should not degrade the performance of both the requestor and provider protocols. For these reasons, it advisable to pre-fetch and cache exported data (when possible), serving a series of accesses to the same abstraction with fewer callback executions. However, this approach also requires the presence of cache invalidation mechanisms, which protocols can use to stale pre-fetched or cached abstractions.

As presented here, the NeSt should come with an *a priori* set of abstractions for data and events, to which protocols adapt in order to cross-interact. A more mature and desirable approach would reverse the adaptation process, having the vertical component adapt to whatever the protocols provide. For example, this adaptation issue could be solved through the use of *reflection*, a characteristic of some modern programming languages[12] that enables introspection of software components, allowing for dynamic changes in behavior. A NeSt reflective API would allow each protocol to define its contribution to cross-layer interactions, providing an initial registration of profiles describing the data and the events it is able to share. The resulting data and event sharing would be more *content-based* than the presented *subject-based* mechanism. With this approach, the sole agreement between the two parties would regard the representation of protocol profiles. A solution could be the usage of a language that provides rules to define both profiles data and metadata, as for example the eXtensible Markup Language (XML). This solution would restrict the agreement on the set of tags (i.e., the *grammar*) to use in building profiles. Note that such use of higher-level programming languages would interest only initial negotiation phases between the NeSt and the protocols, without affecting the runtime performance.

One might argue that the NeSt exhibits some conceptual similarities with a management information base (MIB). An MIB is a collection of network-management information that can be accessed, for example, through the Simple Network Management Protocol (SNMP). SNMP facilitates the exchange of information between network devices and enables network administrators to manage performances, find and solve problems, and plan for network growth. Some NeSt functionalities could be realized through a local MIB (storing protocols information), to which other protocols can access in order to read and write data. However, the NeSt and MIBs target different goals. MIBs are designed for network statistics and *remote* management purposes, while the NeSt aims at overall *local* performance improvements. Furthermore, the MIB's nature makes it unsuitable for the real-time tasks typical of NeSt optimizations, which require only local accesses and fine-grained time scales (e.g., in the order of single packets sent/received).

1.3 The Need for Global Evaluation

The NeSt architecture, as described in previous sections, is a *full* cross-layer approach where protocols become adaptive to both application and underlying network conditions. Such an approach brings the stack as a whole to the best operating trade-off. This has been highlighted by Goldsmith and Wicker,[13] where the authors point at global system requirements, like energy saving and mobility management, as design guidelines for a joint optimization. Our approach opens up different perspectives in the evaluation of network protocols. We claim that in a full cross-layer framework, the performance of a protocol should not only be evaluated by looking at its particular functionalities, but also by studying its contribution in cross-layer activities. Therefore, a stack designed to exploit joint optimizations might outperform a "team" of individually optimized protocols.

To give an example, let us consider ad hoc routing, which is responsible for finding a route toward a destination in order to forward packets. With reference to the classifications reported by Royer and Toh[14] and Chlamtac et al.,[15] the main classes of routing protocols are proactive and reactive. While reactive protocols establish routes only toward destinations that are in use, proactive approaches compute all the possible routes, even if they are not (and eventually will never be) in use. Typically, reactive approaches represent the best option: they minimize flooding, computing and maintaining only indispensable routes (even if they incur an initial delay for any new session to a new destination). But what happens when we consider the cross-layer contribution that a routing protocol might introduce in a NeSt framework?

To answer this question, we provide an example of cross-layer interaction between a routing protocol and middleware platform for building overlay networks, where the former contributes exporting the locally collected knowledge of the network topology. Building an overlay network mainly consists of discovering service peers, and establishing and maintaining routes toward them, as they will constitute the backbone of a distributed service. The overlay network is normally constituted by a subset of the network nodes, and a connection between two peers exists when a route in the underlay (or physical) network can be established. The task of building and maintaining an overlay is carried out at the middleware layer, with a cost that is proportional to the dynamics of the physical network. Overlay platforms for the fixed Internet assume no knowledge of the physical topology, and each peer collects information about the overlay structure in a distributed manner. This is possible because the fixed network offers enough stability, in terms of topology, and bandwidth to exchange messages. Of course, similar conditions do not apply for ad hoc environments, where bandwidth is a precious (and scarce) resource and the topology is dynamic. In ad hoc networks, cross-layering can be exploited, offering the information exported by the network routing to the middleware layer. The key idea is that most of the overlay management can be simplified (and eventually avoided) on the basis of already available topology information.[16] In this case, the more information available, the easier the overlay management; and for this reason, a proactive routing approach becomes more appealing. To support this claim, let us look at what is described by Schollmeier et al.[5] This article describes a cross-layer interaction between a middleware that builds an overlay for peer-to-peer computing and a dynamic source routing (DSR) at the network layer. In this work, the DSR algorithm is forced to maintain valid routes toward the overlay peers, even if these routes are not in use. That is, a reactive routing is forced to behave proactively, with the additional overhead of reactive control packets. The same cross-layer approach with a proactive protocol would probably represent the best joint optimization.

Another example of joint optimization is the extension of routing to support service discovery. A service discovery protocol works at the middleware layer to find out what kind of services are available in the network. As the dynamics of the ad hoc environment determines frequent changes in both available services and hosting devices, service discovery is of fundamental support. The IETF proposes the Service Location Protocol (SLP)[17] to realize service discovery in both Internet and ad hoc networks. Recently, they also underlined the similarity of the messages exchanged in SLP, with those used in a reactive routing such as the Ad hoc On-demand Distance Vector (AODV) protocol.[18] This proposal discusses an extension of AODV to allow service request/reply messages in conjunction with route request/reply. In this proposal, there is a background cross-layer interaction that allows SLP to interface directly with AODV, asking for service-related messages, providing local service data, and receiving service information coming from other nodes. The proposed joint optimization would work even better in the case of a proactive routing protocol such as DSDV or OLSR (see Chlamtac et al.[15] for details). In case of proactive routing, the service information regarding the local services offered on each node could be *piggybacked* on routing control packets and proactively spread around the network, together with local connectivity information. The service discovery communication could be significantly reduced, at the expense of broadcasting routing control packets a few bytes longer. This optimization would result in a proactive service discovery, where a component such as the NeSt supports the exchange of service information from the service discovery protocol, at the middleware, with the routing protocol, and vice versa.

1.4 Discussion and Conclusions

Typically, cross-layering is emphasized as a way to work around the TCP/IP implementation limits, as it introduces direct interaction between protocols to enable smarter adaptation or better performance, at the cost of a "spaghetti-like" code.[9] We believe that cross-layering is possible while keeping the layer separation principle, and that the Internet community is incrementally moving toward cross-layering. The simplest step in this direction is represented by layer triggers, which are predefined signals that notify events between protocols. An example is given by the Explicit Congestion Notification (ECN) mechanism, which notifies the TCP layer about congestion detected by intermediate routers (IP layer). In this way, the source can be informed of congestion quickly and unambiguously, without resorting to inferring mechanisms based on retransmit timer or repeated duplicate ACKs. Another example is given by L2 triggers,[19] added between the data link and IP layers to efficiently detect changes in the wireless links' status. A further step toward cross-layering is presented in by Waldvogel and Rinaldi.[20] This work proposes a way to build topology-aware overlay networks, where logical neighbors are also close in the physical network, according to metrics coming from different levels. These include metrics typically used in routing protocols, such as the physical distance and the bandwidth achieved by a TCP stream.

An open question is to understand if a NeSt-like approach can support cross-layering in the future Internet architecture. Considering the above examples, the NeSt could easily handle the described interactions in the following way:

- The signaling of the ECN bit might correspond to an internal event generated by the IP protocol, previously subscribed by the TCP. Analogously, the MAC layer could generate link-related events to notify the IP layer.
- The metrics used in the construction of the topology-aware overlay network could be associated to NeSt abstractions, seized by the routing and transport protocols, and accessible through the NeSt API.

The NeSt could also be employed to realize optimizations, proposed for the Internet architecture, which are not cross-layered but sidestep the standard layer interfacing. For example, Application Level Framing (ALF)[21] aims at minimizing retransmissions due to data loss, enabling the application level to break the data (to transmit) into suitable aggregates, and the lower level to preserve these frame boundaries when processing the data. Thus, the data segmentation functionality is moved from the transport layer to the application layer. Although the vertical data pipeline inside the protocol stack is not altered, the ALF approach needs a customized implementation, which complicates the maintainability of the overall stack and disagrees with standard interfacing. In the NeSt architecture, the same goal could be achieved keeping the data segmentation functionality at the transport layer and allowing the application layer to instruct, through information sharing, the transport protocol on the way to break data.

Finally, cross-layering provides an effective step toward the mobile Internet. The NeSt architecture is a building block for context-aware computing and networking, a novel paradigm in which a system shows the ability to discover and take advantage of contextual information. Context can be defined as the set of environmental states and settings that determines a system's behavior, and in which system events of interest for the user occurs.[22] While current distributed Internet applications and middleware platforms tend to provide a transparent representation of the underlying execution environment,[23] context awareness fits well in the area of mobile computing, where applications and software components have to cope with dynamic environments, determining changes in context. In mobile networks, applications and middleware platforms need to be aware of context details, leaking out information such as device location, network bandwidth, or surrounding environment, to and from adjacent layers. The NeSt approach goes toward full awareness of networking context: if applications and network protocols are mutually aware of their operating conditions, then they can adjust their behavior to achieve functional trade-offs and deliver the best end-to-end performance.[13]

Awareness is an important requisite for extending the mobile Internet toward 4th Generation wireless technology. The computing world is experiencing a seamless integration of mobile ad hoc networks

with other wireless networks and the fixed Internet infrastructure. The global system presents different characteristics, depending on both physical constraints (e.g., bandwidth, energy, processing power) and usage patterns. The key requirement for the operation of this *heterogeneous* Internet is the protocol's ability to globally adapt to application requirements and underlying network conditions. The need for adaptive networking becomes a challenging issue, the solution of which requires context-awareness.

Acknowledgments

This work was partially funded by the Information Society Technologies programme of the European Commission, Future and Emerging Technologies, under the IST-2001-38113 MOBILEMAN project, and by the Italian Ministry for Education and Scientific Research in the framework of the FIRB-VICOM project.

References

1. J.P. Macker and M.S. Corson. Mobile ad hoc networking and the IETF. *ACM Mobile Computing and Communications Review,* 2(3):7–9, 1998.
2. M.S. Corson, J.P. Macker, and G.H. Cirincione. Internet-based mobile ad hoc networking. *IEEE Internet Computing,* 3(4):63–70, 1999.
3. C. Szyperski. *Component Software,* pp. 140–141. Addison-Wesley, 1998.
4. K. Chen, S.H. Shah, and K. Nahrstedt. Cross-layer design for data accessibility in mobile ad hoc networks. *Wireless Personal Communications,* 21(1):49–76, 2002.
5. R. Schollmeier, I. Gruber, and F. Niethammer. Protocol for peer-to-peer networking in mobile environments. In *Proc. 12th IEEE Int. Conf. Computer Communications and Networks,* Dallas, TX, 2003.
6. W.H. Yuen, H. Lee, and T.D. Andersen. A simple and effective cross layer networking system for mobile ad hoc networks. In *Proc. IEEE PIMRC 2002,* Lisbon, Portugal, 2002.
7. M. Chiang. To Layer or not to layer: balancing transport and physical layers in wireless multihop networks. In *Proc. IEEE INFOCOM 2004,* Hong Kong, China, 2004.
8. U.C. Kozat, I. Koutsopoulus, and L. Tassiulas. A framework for cross-layer design of energy-efficient communication with QoS provisioning in multi-hop wireless netwroks. In *Proc. IEEE INFOCOM 2004,* Hong Kong, China, 2004.
9. V. Kawadia and P.R. Kumar. A Cautionary Perspective on Cross Layer Design. In *IEEE Wireless Communications,* 12(2):3–11, 2005.
10. M. Conti, S. Giordano, G. Maselli, and G. Turi. MobileMAN: mobile metropolitan ad hoc networks. In *Proc. 8th IFIP-TC6 Int. Conf. on Personal Wireless Communications,* pp. 169–174, Venice, Italy, 2003.
11. M. Conti, G. Maselli, G. Turi, and S. Giordano. Cross-layering in mobile ad hoc network design. *IEEE Computer, Special Issue on Ad Hoc Networks,* 37(2):48–51, 2004.
12. Sun Microsystems. The JAVA Reflection API. http://java.sun.com/j2se/1.4.2/docs/guide/reflection/index.html, 2002.
13. A.J. Goldsmith and S.B. Wicker. Design challenges for energy-constrained ad hoc wireless networks. *IEEE Wireless Communication,* 9(4):8–27, 2002.
14. E.M. Royer and C.-K. Toh. A review of current routing protocols for ad hoc mobile wireless networks. *IEEE Wireless Communications,* 6(2):46–55, 1999.
15. I. Chlamtac, M. Conti, and J.J.-N. Liu. Mobile ad hoc networking: imperatives and challenges. *Ad Hoc Networks Journal,* 1(1):13–64, 2003.
16. M. Conti, E. Gregori, and G. Turi. Towards scalable P2P computing for mobile ad hoc networks. In *Proc. First Int. Workshop on Mobile Peer-to-Peer Computing (MP2P'04), in conjunction with IEEE PerCom 2004,* Orlando, FL, 2004.
17. E. Guttman, C.E. Perkins, J. Veizades, and M. Day. Service Location Protocol, Version 2. IETF RFC 2608, June 1999.
18. R. Koodli and C.E. Perkins. Service Discovery in On-Demand Ad Hoc Networks. Internet Draft, October 2002.

19. S. Corson. A Triggered Interface. http://www.flarion.com/products/drafts/draft-corson-triggered-00.txt, May 2002.
20. M. Waldvogel and R. Rinaldi. Efficient topology-aware overlay network. *ACM Computer Commun. Rev.,* 33(1):101–106, 2003.
21. D.D. Clark and D.L. Tennenhouse. Architectural considerations for a new generation of protocols. In *Proc. ACM Symp. Communications Architectures and Protocols,* pp. 200–208. ACM Press, 1990.
22. G. Chen and D. Kotz. A Survey of Context-Aware Mobile Computing Research. Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, 2000.
23. C. Mascolo, L. Capra, and W. Emmerich. Middleware for mobile computing (a survey). In E. Gregori, G. Anastasi, and S. Basagni, Editors, *Neworking 2002 Tutorial Papers,* LNCS 2497, pp. 20–58, 2002.

2

Routing Scalability in MANETs

Jakob Eriksson,
Srikanth Krishnamurthy
Michalis Faloutsos

With today's rapidly improving link-layer technology, and the widespread adoption of wireless networking, the creation of large-scale ad hoc networks could be construed as all but inevitable. However, for routing in such a network to be feasible, there is a pressing need for a scalable ad hoc routing protocol. Applications for large-scale ad hoc networking include consumer-owned networks, tactical military networks, natural disaster recovery services, and vehicular networks.

Ad hoc routing protocols used experimentally today, such as DSDV, OLSR, AODV, and DSR, only scale reasonably well to dozens or sometimes hundreds of nodes. To support networks one or several orders of magnitude larger, there is a need for routing protocols designed specifically to scale to large networks. Under certain limiting assumptions, geographical location information can be used to help the routing layer scale to support very large networks. However, this chapter focuses on the more generally viable approach of multilevel clustering, which to some extent is what has made made the Internet scale as well as it does.

We study various aspects of routing protocol scalability. First, we take a look at the analytical results thus far, with regard to ad hoc network scalability. These results assess the theoretical limits for ad hoc network scalability in terms of the capacity achievable per node in the network. To set the stage for the

scalable routing techniques, and to introduce the reader to the issues that impact scalability, we briefly discuss several techniques used for ad hoc routing. These include flat proactive routing, pure reactive routing, geographic routing, and zone-based hybrid protocols. We then take a more detailed look at routing based on clustering, in its single-level and multilevel forms. Finally, we spend the last third of the chapter describing a recent promising scalable routing technique based on multilevel clustering, called Dynamic Address Routing.

2.1 Defining Scalability

The scalability of a network protocol can potentially be defined in many different ways, and at several different levels. In this chapter, we use the following high-level definition of scalability.

> **Scalability** *is the ability of a routing protocol to perform efficiently as one or more inherent parameters of the network grow to be large in value.*

Typical parameters that are studied for ad hoc networks are the number of **nodes (N)** and the average rate of **mobility(M)** in m/s under various mobility models. Other parameters that have an impact on scalability include **node density (D)**, number of **links (L)**, the **frequency of connection establishment (F)**, and the average number of **concurrent connections (C)**. Measuring performance can also be done in several ways. Typical metrics used to evaluate routing protocols are overall message or byte overhead, amount of per-node state to be maintained, latency, and total network throughput. In this chapter, we primarily discuss the overhead aspect. However, we also discuss the other metrics briefly in the sections that follow.

In the remainder of this chapter, we use notation commonly employed in asymptotic analysis to describe various scalability characteristics. In particular, we use the $\Omega(X)$ to denote a lower asymptotic bound, $O(X)$ to denote an upper asymptotic bound, and $\Theta(X)$ to denote a simultaneous upper *and* lower asymptotic bound. By asymptotic bound, we refer to the scaling behavior of the protocol with respect to a given variable. For example, if a protocol is said to have an overhead of $O(N)$, this means that there exists a constant c such that the amount of overhead incurred in a network of N nodes is at most cN, where N can take on any finite value. Except where explicitly stated, node identifiers are taken to be 48-bit MAC addresses. It is reasonable to assume that a 48-bit identifier space will not be exhausted within the foreseeable future ($2^{48} = 281,474,976,710,656$ or about 281 trillion unique identifiers).

2.2 Analytical Results on Ad Hoc Network Scalability

The analytical study of scalability relationships in ad hoc networks can provide us with valuable insights into the proper design of ad hoc routing protocols and possibly related mechanisms at other layers. So far, the study of scalability in ad hoc networks has been mostly limited to simulation. However, a few significant analytical results have emerged fairly recently, and we introduce them in this section.

2.2.1 Link Layer

Even without considering the effects of routing overhead on the performance of ad hoc networks, there are several concerns regarding the scalability of current wireless networking link-layer technology.

It is easily seen that the popular 802.11 link layer, when deployed with omnidirectional antennas, does not scale with respect to node density, D. Clearly, as D grows, each node will receive only a proportional share of the channel capacity. The upper limit on the average link layer capacity made available to each node decreases as $1/D$. A well-known solution to this problem is to reduce the transmission range of each node, thereby reducing D. The effect achieved is called *spatial reuse*, where several transmissions can take place on the same frequency band simultaneously, due to the limited spatial overlap of the transmitters involved.

However, as a direct effect of reducing the transmission range, packets in some cases must be forwarded over an increased number of wireless links to reach their respective destinations. Increasing the number of hops is likely to lead to longer end-to-end delays, lower packet delivery ratios, and in some cases, increased traffic congestion.

A fundamental result in multihop ad hoc networking was shown by Gupta and Kumar.[11] A simplified argument for their result follows. In a network of nodes with omnidirectional antennas, and with a constant node density, we can expect the average path length to be $\Theta(\sqrt{N})$, where N is the number of nodes in the network. Therefore, for every packet a node generates, it will see, on average, $\Theta(\sqrt{N})$ packets originated by other nodes. Thus, with a channel capacity of C, the capacity available for a node's own packets will be:

$$O\left(\frac{C}{\sqrt{N}}\right) \tag{2.1}$$

where C is the total channel capacity, that is, the maximum throughput achievable by a single link when there are no other links competing for the channel. The unfortunate conclusion is that under certain reasonable assumptions, purely omnidirectional ad hoc networks cannot grow beyond certain fairly restrictive limits. However, we would like to point out that all hope is not lost. As link layer technologies evolve, the channel capacity C will continue to increase. And for every increase in channel capacity, the feasible network size grows by the square of this increase, as per Equation 2.1. Since the publication of the article by Gupta and Kumar,[11] channel capacity has grown by approximately 100 times. Our conclusion is that whatever the feasible network size was at the time of publication (1999), the upper limit today is 10,000 times higher. Clearly, this shows that link layer capacity by itself is not the limiting factor in multihop ad hoc networks. Note that this highly theoretical result does not take into account any routing layer overhead, the scalability of which is the topic of this chapter.

In addition, there is the prospect of using directional antennas[16] and adaptive beamforming antennas.[26] These could be employed to have nodes dynamically direct a narrow transmission beam toward the neighbor it wishes to communicate with, thereby greatly improving both transmission range and spatial reuse.*

Grossglauser and Tse[10] published a somewhat controversial result. The authors show that if nodes are mobile, then each node could potentially achieve a throughput that *does not* decrease with the size of the network. By relaying each packet only once, to a random one-hop neighbor, a source can achieve a stationary uniform distribution of its packets throughout the network. Subsequently, as the destination moves around, each of its neighbors will always have packets to deliver to it. As each packet only traverses two hops, the throughput of the node can be expected to remain the same, regardless of the size of the network.

This result relies on strong assumptions with regard to the mobility patterns of the nodes, and even given those assumptions, the expected delay is of the order of node mobility, in the sense that nodes have to move considerable distances before a packet can be delivered. In our opinion, although this result holds in theory, it is unclear as yet if it will have much practical relevance.

2.2.2 Hierarchical Routing

Hierarchical routing protocols, such as those based on multilevel clustering, consist of a number of different components, such as clustering, routing, and location management. Here, clustering is the process by which nearby nodes form groups, called *clusters*. For the purpose of routing, clusters can be treated as a single destination, thereby reducing the amount of routing state that must be maintained at each node. Location management is any technique by which a source can determine the current address or *location* of an intended destination node, given its identifier.

When studying the scalability of such protocols, the scaling properties of each of these components must be considered. Sucec and Marsic[29,30] have studied the theoretical scalability aspects of multilevel hierarchical

*To see how both range and spatial reuse can be improved simultaneously, consider the extreme case of directional transmission, a point-to-point laser link.

routing in ad hoc networks. In the general scheme they analyze, nodes are organized in clusters, which are then grouped in higher-level clusters. The number of levels is logarithmic in the network size. The location management technique they analyze is a distributed location server, where each node stores the current address of $\Theta(\log N)$ other nodes, where N is the number of nodes in the network. A similar location management scheme is discussed in Section 2.9. Specifically, their analysis focuses on the number of routing-related control datagrams that a node needs to transmit per unit of time, on average, given a wide variety of parameters.

Their main result is that routing overhead is polylogarithmic in the size of the network. More specifically, the channel capacity required for routing control messages sent by each node, on average, is:

$$\Omega(\log^3 N)$$

Interestingly, they show that the dominating factor in the overhead calculation is not routing updates, but the retransmission of location information due to changes in the clustering hierarchy, called *location management handoff*. Other potentially valuable results of the same article include the overhead incurred by cluster formation and maintenance, which is computed to be

$$O(\log\ N)$$

packet transmissions per node per second, and the overhead for location management handoff, which is shown to be

$$\Theta(\log^2 N)$$

packet transmissions per node per second, where the size of every control packet is $\Theta(\log\ N)$. Note that this study is targeted at a particular group of clustering schemes (Max-Min D-hop clustering[1]). These are based on finding the node with the maximum node identifier in a D-hop neighborhood, and assigning that node to be the cluster head. Other types of clustering, such as that described in Section 2.9, could potentially have different scaling behaviors. It is also geared toward a scalable hierarchical location management scheme similar to that used by Eriksson et al.[7,8] and described in Section 2.9. Again, other types of location management will exhibit different scaling behavior. Nevertheless, these results offer valuable insights into the scalability of hierarchical, multilevel clustering ad hoc routing protocols. To our knowledge,[30] is the first paper with comprehensive theoretical results on the overhead of multilevel hierarchical routing protocols.

2.3 Flat Proactive Routing

Flat proactive routing scales very well with respect to the frequency of connection establishment (F) and the number of concurrent connections (C). However, the number of control packet transmissions per node is $\Theta(N)$.

In proactive routing, the routing protocol periodically disseminates routing information throughout the network. With flat proactive routing, every node keeps routing information for every other node; there is no abstraction for nodes far away. This strategy generally leads to close to optimal paths, but this is achieved at the cost of lacking scalability. The flat proactive routing protocols proposed so far can be roughly divided into two subcategories: (1) link-state (LS) and (2) distributed Bellman-Ford (DBF) algorithms.

In LS algorithms such as Fisheye State Routing,[22] Global State Routing,[4] and Optimized Link-State Routing,[6] each node has complete, although not always accurate, knowledge of the state of every link in the network. Using this information, it can calculate the entire path to the destination on its own accord. This has many advantages. In particular, recovery from link failure is typically very quick in LS protocols. With large N or D, the number of links in the network, and thus the routing table size, may be prohibitive. Fisheye State Routing (FSR)[22] tries to reduce the overall overhead by limiting the rate of link-state updates far away from the source of the update. The idea in FSR is that link changes far away generally have a small effect on local routing decisions.

In DBF algorithms, such as Destination Sequenced Distance Vector routing[24] and Wireless Routing Protocol,[19] each node has much less information about the network. For every destination, a node maintains

a routing table consisting of the distance to the destination, and the next hop neighbor on the shortest route toward the destination. Typically, after a link failure, there is an interval of time where faulty routes may exist, until the protocol has settled on a new route.

Common for all of these protocols is that the necessary amount of state kept at each node scales at least linearly with N. In a mobile network, this state must be updated frequently, resulting in protocol overhead on the order of $O(N)$.[30] For this reason, flat proactive routing protocols are only feasible for small networks.

2.4 Pure Reactive Routing

Reactive routing is scalable with respect to most parameters, as long as the frequency of connection establishment (F), and the average number of concurrent connections (C), remain low. Control packet transmissions per node grow as $O(F + C)$, which is $\Omega(1)$, but $O(N^2)$.

In an effort to address the problem of maintaining state for all nodes in the network, reactive protocols such as Ad hoc On-demand Distance Vector routing,[25] Dynamic Source Routing,[13] Associativity Based Routing,[31] and Labeled Distance Routing[9] defer the expenditure of routing overhead until the time of connection establishment. With this technique, nodes keep completely quiet as long as there is no data to transmit. If a connection is to be established, the source node S needs to flood the network with a route request, as shown in Figure 2.1. When the intended destination D receives the route request, it responds to the source with a route response, using one of the routes discovered during the route request phase. In networks where a large majority of nodes have nothing to send, and where connections involve more than just a few packets, this strategy pays off in terms of reducing the overall routing overhead.

Reactive routing protocols have seen much popularity in ad hoc networks research. This is due to several good reasons, including the battery savings achieved by not transmitting anything during idle periods. Other important reasons are the good performance and the straightforward design principles of AODV and DSR, the two most well-known reactive ad hoc routing protocols.

However, by deferring the routing overhead, these protocols lose many potential aggregation benefits made possible by proactively distributing routing information. In contrast with flat proactive routing, every connection establishment sets off a reactive route request with an asymptotic cost of $O(N)$, as a nonnegligible constant fraction of all nodes will rebroadcast the request packet. In addition, in a mobile network, established connections will fail regularly due to link breakages caused by node motion, thereby initiating additional route requests. This gives an overhead complexity of $\Theta(F+C)$ for the number of route requests per second. Putting the two terms together, the expected number of control packet transmissions is $O(N(F+C))$. With N nodes to share the burden, the average per-node cost is $O(F+C)$. Note that if a

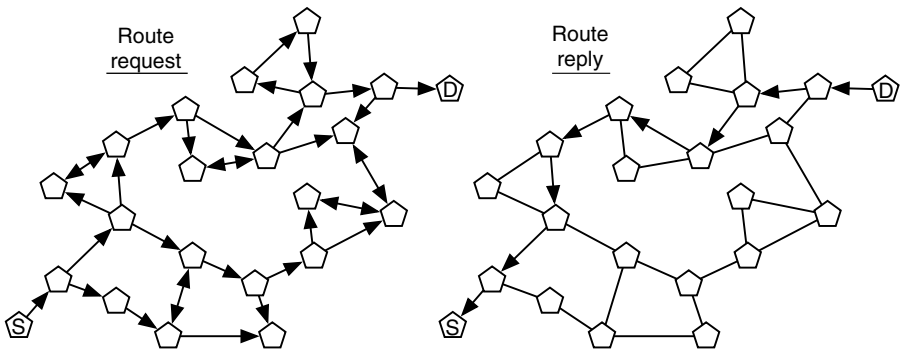


FIGURE 2.1 Reactive routing. A route request is flooded throughout the network. Once the request reaches the intended destination, a route reply is sent back along a discovered path.

constant fraction of the nodes can be expected to start or maintain a connection every second, this reverts back to the $O(N)$ per-node cost of flat proactive routing. In the worst case, where a constant fraction of the nodes can be expected to set up k connections, and k grows linearly with N, the overhead incurred will be $O(N^2)$.

A performance optimization used aggressively in DSR[13] is route caching, where intermediate nodes are allowed to send a route response, if they have recently observed a route to the desired destination. This can result in greatly improved performance but there is also a high risk of *route poisoning*, where intermediate nodes unwittingly return routes that are no longer accurate. In general, reactive routing has been shown through simulation to scale better than flat proactive routing in most considered scenarios.

As we see below, proactive routing has an advantage that a purely reactive protocol lacks: the ability to cluster nodes and aggregate routes. As we see in the following sections, clustering and address aggregation have the potential to drastically reduce the protocol overhead of a proactive routing protocol, as network size increases. The relationship between the overhead of reactive and proactive routing under different traffic scenarios is discussed in more detail by Eriksson et al.[8]

2.5 Geographical Routing

The control overhead of geographical routing is typically $O(1)$, not counting location management. However, geographical routing relies heavily on two assumptions: (1) that each node knows its position, and (2) that the geographical distance between nodes corresponds well to the distance between these nodes in the network topology. In many situations, these assumptions are unacceptable.

Geographical routing protocols make use of the geographical location of a node to make routing decisions. Such location information would generally be acquired either from GPS satellites, or from location interpolation given the positions of neighboring nodes.

In addition to knowing its own geographical location, a node also needs to know the locations of its neighbors, as well as the location of its intended destination. Dream (Distance Routing Effect Algorithm for Mobility)[2] and Grid Location Service[17] are mechanisms for finding out the location of any given node in the network. In Dream, nodes periodically flood their location information throughout the network. However, as the flood travels away from the source, the speed with which updates are propagated decreases, thereby drastically reducing the overall overhead of the protocol. This is similar to the technique used in Fisheye routing[22] to reduce the cost of disseminating link state updates. In GLS, the location for a given node is stored at an *anchor node*. An anchor node is defined as the node positioned closest to a geographic location that is determined by hashing the node identifier. Every node is responsible for keeping its anchor node up-to-date on its current location. This method of distributing responsibility for storing location information is highly scalable and efficient, given that some characteristics of the network are known, such as the extent of the network in geographical terms.

Geographic routing protocols include Greedy Perimeter State-less Routing (GPSR)[14] and Location Aided Routing (LAR).[15] GPSR greedily routes packets to the one-hop neighbor that is closest to the destination. Should an obstacle appear between source and destination, GPSR uses a planarized version of the network graph and follows the "right hand rule" to route around the obstacle. The technique is illustrated in Figure 2.2, where a packet destined for node D is originated at S. When the large obstacle in the middle of the network is encountered, the *right-hand rule* is triggered, routing the packet around it. The use of the *right-hand rule* for routing around obstacles can result in paths of length $O(N)$, making greedy routing a risky proposition unless the characteristics of the topology are known in advance. LAR uses the geographic location of the destination to guide a reactive route lookup. By limiting the route request flood to neighbors in the approximate direction of the destination, the cost of route setup is reduced.

Any geographical routing protocol relies on the assumption that the geographical distance between two nodes corresponds well with their distance in the network topology. In scenarios where this is not the

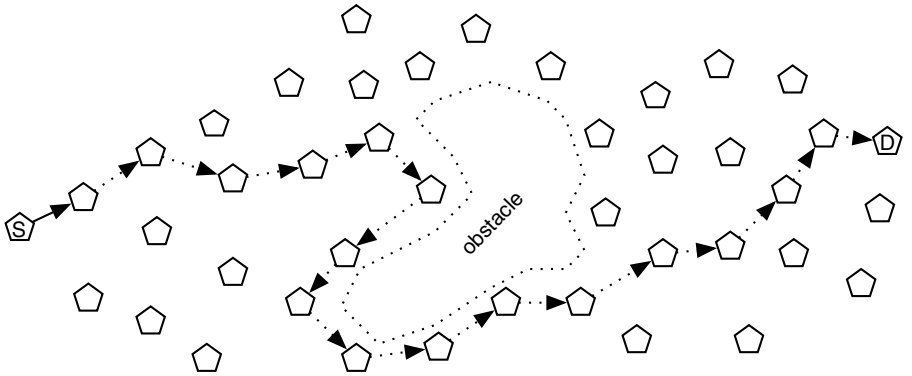


FIGURE 2.2 Geographical routing. The next hop is selected on greedily, until there is no neighbor that is closer to the destination than the current node. When this happens, routing switches to the *right-hand rule* until the obstacle has been successfully routed around.

case, such as sparse or heterogeneous networks, or networks with directional or wired links, geographical routing is unlikely to achieve acceptable performance.

2.6 Zone-Based Routing

Zone-based routing combines the merits of flat proactive and pure reactive routing. However, while these hybrid protocols are more efficient than the component protocols they are made up out of, the asymptotic scalability of zone-based routing is the same as that of other flat routing protocols.

In the Zone Routing Protocol (ZRP)[12] and Sharp Hybrid Adaptive Routing Protocol (SHARP),[28] the merits of proactive and reactive routing are combined to form two hybrid proactive–reactive protocols. Both protocols follow a similar architecture. Around every node, a *zone* of d hops is maintained in which proactive routing is performed. For all destinations outside the zone, reactive route requests are used to establish a route. As soon as a route request reaches a node in the zone of the intended destination, this node replies with a route response.

In ZRP, the size of the zones can be varied, depending on the mobility and traffic characteristics of the network.[20] Thanks to the proactive routing information available within the zone, the damaging effect of the flood is limited, as route requests can be efficiently routed to the edges of the zone, using a technique called *bordercasting*. Several other techniques are introduced to minimize the duplication of effort that could otherwise happen due to zone overlap.

While ZRP introduces its own routing components, such as *bordercasting*, SHARP is a straightforward combination of proactive and reactive routing (Figure 2.3). In SHARP, every node individually adapts the size of its zone, that is, the distance (in hops) up to which its proactive routing updates should be forwarded. Reactive routing is done according to whatever reactive protocol is used, with the modification that intermediate nodes that have proactive routing information for the desired destination node are allowed to reply to the route request. SHARP trades off the constant overhead of proactive routing against the high incremental cost of reactive routing by adaptively tuning the zone size of a node to correspond to the popularity or usage profile of the node. In addition to improving performance for popular nodes, the same trade-off is used to achieve desired packet delivery ratio and delay characteristics.

However, if properly done, route caching in reactive routing protocols can likely achieve a constant-term savings in terms of protocol overhead. Moreover, neither route caching nor the hybrid approaches taken in ZRP and SHARP can efficiently handle the case where there are frequent connection establishments, unless traffic is concentrated to a small number of nodes. For SHARP to achieve a successful trade-off

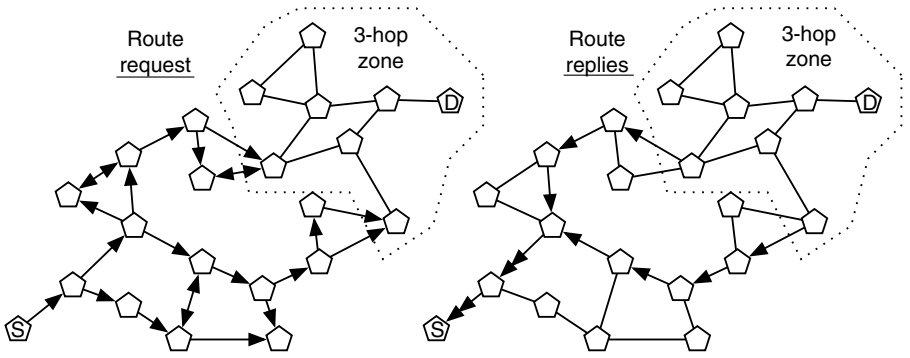


FIGURE 2.3 Hybrid proactive–reactive routing with SHARP. Route requests are flooded until they reach a node within the destination's proactive zone.

between flat proactive and reactive routing, it is necessary for a few nodes to receive the majority of the network traffic.

Compared to flat proactive routing, or pure reactive routing, this middle ground between reactive and proactive routing can be expected to achieve lower overhead and delay under many traffic scenarios. However, although hybrid methods can be expected to reduce routing overhead, they only do so by a constant factor.

2.7 Single-Level Clustering

Single-level clustering improves scalability with respect to the network size (N) if the size of each cluster can be set to $\sqrt{(N)}$. *In this case, the control packet overhead is* $\Theta(\sqrt{N})$. *With constant size clusters, overhead remains at* $\Theta(N)$. *Certain node mobility patterns (M) can have a larger detrimental effect on the performance of clustering protocols than they have on flat protocols.*

Several protocols propose to use clustering to improve routing protocol scalability. *Clustering* is a process by which neighboring nodes form connected subsets, with one node elected as the cluster head. Depending on the clustering technique used, clusters can be of radius of one or more hops from the cluster head. The cluster heads may have responsibilities in addition to that of a regular node, such as inter-cluster routing and intra-cluster coordination.

In hierarchical protocols, such as LANMAR[21] and CGSR,[5] routes are aggregated by cluster. Inside a cluster, every node has complete routing information for every other node in the cluster. Externally, however, only a route to the cluster as a whole is published. Packets are first routed toward the cluster head of the destination. Once the cluster head, or simply any node within the destination cluster, has been reached, the packet is routed directly toward its final destination within the cluster. Through this technique, a smaller amount of routing state is necessary on each node, and intra-cluster changes in the topology do not affect external routes. Note that all routing schemes that use clustering for routing will incur a cost in terms of increased path length. However, this cost is usually negligible compared to the savings achieved by reducing the amount of routing overhead incurred.

In contrast to the hybrid schemes mentioned earlier, which rely on flooding, hierarchical schemes also need to keep track of which cluster a node belongs to. This is sometimes referred to as *location management*. Depending on the assumptions used, location management can be a crucial factor in the performance of a clustering-based routing protocol.

In LANMAR (Figure 2.4), it is assumed that most nodes will remain in the same cluster throughout their lifetimes, and group membership is determined at network initialization. The authors use a *group mobility* model, which applies mostly to military scenarios. LANMAR builds on ideas from Landmark Routing[32]

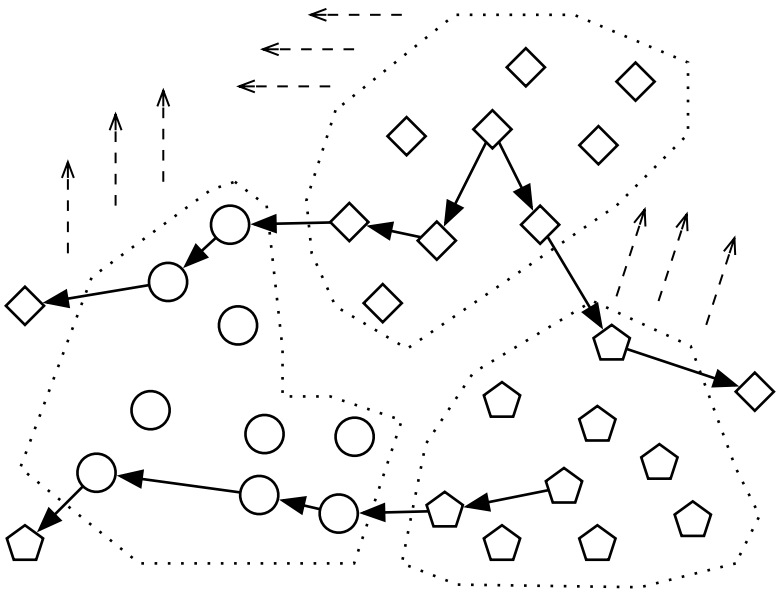


FIGURE 2.4 Mobile groups and stray nodes in LANMAR. Nodes move together in groups, while stray nodes are handled with separate distance vector entries.

and Fisheye State Routing (FSR).[22] Nodes within a cluster exchange link-state information using FSR. In addition to this link-state information, each node keeps a distance vector table for a specific node in each cluster. This node is referred to as the Landmark. Any stray nodes, that is, nodes that are not directly connected to their home cluster, are handled as special cases: a separate distance vector routing entry is kept by every node on the shortest path between the Landmark node and the stray node. Assuming that only a constant number of nodes stray from their home clusters, the asymptotic control packet overhead is $\Theta(\sqrt{N})$.

In CGSR,[5] one-hop clustering is performed, and is mainly used for transmission scheduling. A technique is also proposed in which each node globally advertises its cluster membership, and routing entries are kept only for cluster heads. Because the cluster radius is limited to a single hop, a cluster will contain only a constant number of nodes, leading to, at best, a constant improvement in the overhead incurred.

Both of these protocols rely on nodes staying within their original clusters throughout their lifetimes, or overhead will grow quickly. More flexible and scalable location management methods have been developed, and these are discussed in the upcoming sections.

As with the hybrid scheme above, these single-level clustering protocols only reduce overhead to at best $O(\sqrt{N})$, depending on the cluster size. In the next section, we discuss how to extend the idea of cluster-based routing to reduce the size of the routing tables from $O(\sqrt{N})$ to $O(\log N)$.

2.8 Multilevel Clustering

Multilevel clustering protocols scale well with network size (N), frequency of connection establishment (F), and the number of concurrent connections (C). The number of control packet transmissions per node is $\Omega(\log^2 N)$.

For large networks, the address size in bits is $\Omega(\log^2 N)$*, which in practice could easily grow beyond the limit of feasibility.*

The ability to achieve true routing scalability with respect to network size (N), under most common scenarios, has so far only been demonstrated through the use of multilevel clustering. In these protocols,

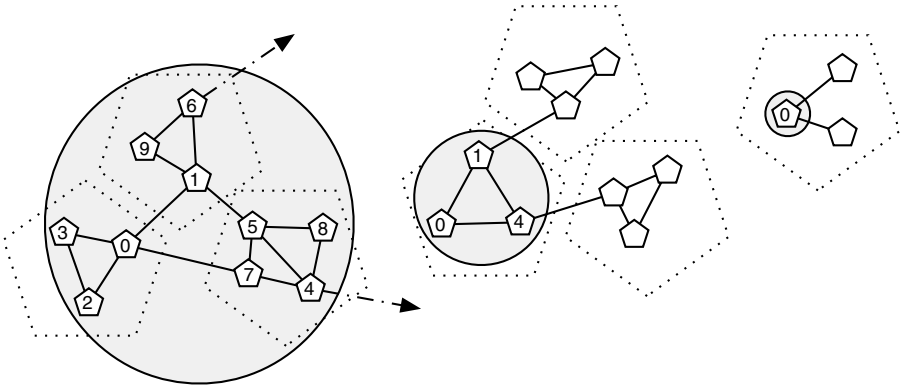


FIGURE 2.5 An example multilevel cluster hierarchy. At the left, individual nodes with their respective node IDs, partial view. In the middle, level-1 clusters forming level-2 clusters; and to the right, a level-2 cluster view of the entire network.

physical nodes cluster first into level-1 clusters. Then, up to d level-1 clusters are further clustered into level-2 clusters, etc. (Figure 2.5). In general, with a *clustering degree* of d, the size of the routing table will be on the order of $O(d \log_d N)$.

In addition to reducing the size of the routing table, multilevel clustering will also make the network appear much less dynamic, as link-state changes within a given cluster generally are not propagated to nodes that are not part of the cluster. This will reduce the overall control packet overhead under node mobility.

Examples of multilevel clustering are Hierarchical State Routing (HSR)[23] and MMWN.[27] These are link-state protocols, and they use the clustering abstraction to define *virtual links* between clusters. Instead of keeping track of all links in the network, a node now only needs to maintain entries for the virtual links going to or from a cluster in which the node is a member, a much smaller number.

Initially, one-hop physical-level clusters are formed, as in the previous section, by electing a cluster head and having nodes in the k-hop neighborhood of that node join the cluster head to form a cluster. To build the next higher clustering level, the cluster heads of neighboring clusters elect a higher-level cluster head from among themselves. Once the cluster hierarchy is formed, each node creates its own hierarchical identifier (HID) by concatenating the identifiers of all the clusterheads from the root of the hierarchy to the node in question. In theory, the size of a node identifier is $\Theta(\log N)$ bits, which results in an asymptotic HID size of $\Theta(\log^2 N)$. However, in practice, node identifiers are typically 48-bit MAC addresses.*

Data packet headers contain their destination HID. Routing is performed one level at a time: first, a packet is routed directly to the lowest-level cluster head in the HID which exists in the current node's routing table. Once this cluster head has been reached, the routing proceeds to the next lower level, as indicated in the HID. Eventually, the intended destination is reached, through the recursive application of this procedure.

Another example is Landmark routing, which is similar to HSR and MMWN but uses a Distributed Bellman-Ford-like routing scheme and does not concentrate traffic to cluster heads to the same extent. First described by Tsuchiya,[32] Landmark routing establishes a set of self-elected Landmark nodes in multiple levels. The main difference in Landmark routing is how the hierarchy is formed. Here, each Landmark periodically broadcasts an advertisement, announcing its presence. Depending on the level k

*In addition to the obvious practical reasons for using MAC addresses, it is worth noting that if a node identifier is to be constant throughout the lifetime of a node, it must be unique not only in the current network, but in every network it could conceivably be part of. The only way to feasibly assign identifiers to ensure this is to assign every node a globally unique ID, which is the sole purpose of the current 48-bit MAC addresses used by network interface cards.

of the Landmark, the advertisement will travel r_k hops. A new node initially assigns itself level 0 and sends an advertisement. If it can hear the advertisement of a level-1 Landmark, it can remain at level 0 and select that Landmark as its parent. If it does not, it cooperates with its level-0 neighbors to elect a new level-1 node. This process is repeated until a small subset of the nodes in the network are level-d Landmarks, where r_d is larger than the diameter of the network. At this point, the Landmark hierarchy is complete.

An interesting difference between Landmark routing other multilevel clustering schemes is that several Landmarks can cover a single node, giving the node several valid addresses. Landmark routing was the basis for LANMAR mentioned in the previous section, and was later extended in L+ routing[3] and Safari routing.[18]

Safari routing[18] is similar in many respects to Landmark routing. Landmark nodes, here called *drums*, self-elect and form a multilevel Landmark hierarchy. One major difference is that the Safari hierarchy does not extend all the way to the physical (node) level. Instead, it extends down to the level of a *fundamental cell*, consisting of approximately 10 to 100 nodes. Inside a fundamental cell, routing is done by Dynamic Source Routing (DSR).[13] Note that this is the opposite of Zone-based routing, where the local scope is handled by proactive routing, and distant nodes are served through reactive route requests. In Safari, the local scope is handled by DSR, and proactive routing is used for computing routes to more distant nodes, to avoid the high cost of long-range reactive route requests. If the size of the fundamental cell is kept constant, the size of the routing table in Safari is $O(\log\ N)$.

Routing based on multilevel clustering, just like single-level clustering and geographical routing, needs a mechanism through which a node can acquire the current location (HID, or Landmark address) of its intended destination. This has been addressed in a variety of ways, including assumptions of group mobility[21] (which makes the problem go away by assuming that nodes stay with their original clusters), flooding,[5] Mobile IP-style home agents,[23] and distributed location servers.[3,7,8,18,27,30] The distributed location server is the most versatile and scalable of these options. Here, the responsibility for storing the current location of a given node is distributed across the network. MMWN uses a combination of a hierarchical organization of location servers together with *paging*, essentially a restricted flood, to find the current location of a node. A different method is used by Chen and Morris,[3] Mohammed et al.,[18] and Susec and Marsic,[30] similar to the *anchor node* idea in GLS.[17] To find the anchor node of a node i, a function $hash(ID_i)$ is computed. For every level, the cluster with the identifier most similar to $hash(ID_i)$ is selected as the cluster to which the anchor node should belong, until eventually a level-0 cluster (a single node) has been reached. This is the *anchor node* that is responsible for storing the current location of node i. A similar $hash(ID_i)$ is computed, and the node with the *routing address* most similar to $hash(ID_i)$ is the anchor node for node i.[7,8] This is discussed in more detail in Section 2.9. In several of these location management schemes, multiple anchor nodes are selected, such that there are many anchor nodes close to the node and fewer anchor nodes far away from it. This improves the scalability of the distributed location server, as local changes and requests only have an effect on local network resources.

Multilevel clustering protocols that depend on hierarchical identifiers (including Landmark addresses) are highly sensitive to changes in the clustering hierarchy: whenever a cluster head gets disconnected or otherwise leaves the cluster, a new cluster head must be elected, and all the nodes within the affected cluster need to update their hierarchical identifiers. In addition, the election of a new cluster head will change the *anchor node* relationships, causing a necessity for a location-handoff mechanism. This has been identified[30] as the dominating component of the total routing overhead of multilevel hierarchical routing protocols. This takes the total number of control packet transmissions per node in routing protocols based on multilevel clustering to $\Omega(\log^2 N)$, where every packet is of length $\Omega(\log\ N)$ bits.

2.9 Dynamic Address Routing

Dynamic address routing is similar to routing based on multilevel clustering but the address size is reduced to $\Omega(\log N)$ from the $\Omega(\log^2 N)$ address size required with the previous multilevel clustering protocols. Dynamic address routing is also less sensitive to node movement than previous multilevel clustering approaches, because its routing addresses are not built up from individual node identifiers.

TABLE 2.1 Routing Table and Address Sizes for a 1024-Node Network Varying d.

d	Routing Table Size	Routing Address Size	Hierarchical ID Size
2	10	10	480
4	15	10	240
16	45	12	144
64	126	12	96
1024	1023	10	48

Note: Address and ID sizes in bits. Changing routing address size is due to rounding to the nearest $\log_2 d$ bit word.

Dynamic address routing, described by Eriksson et al.,[7,8] takes the idea of multilevel clustering one step further. Whereas previous multilevel clustering schemes use cluster identifiers to form addresses (HIDs), dynamic address routing dynamically assigns a considerably shorter *index* to each cluster. The *routing address* of a node is formed by concatenating the *indices* of the cluster that the node belongs to at every level. In more detail, with a clustering degree of d, each of the $1 \ldots d$ clusters belonging to the same higher-level cluster gets an index in the range $0 \ldots d - 1$. The more lengthy cluster identifiers are used only to ensure that there is a single, unique cluster per index.

As shown in Table 2.1, the difference in size between hierarchical identifiers and routing addresses is dramatic. With a low clustering degree d, the size of the hierarchical identifiers can be quite daunting. In contrast, the routing addresses used in dynamic address routing are roughly constant with respect to d.

Moreover, the size of the routing table in a multilevel clustering hierarchy is equal to

$$(d - 1) \log_d N$$

There are $\log_d$ levels and $d - 1$ routing entries per level. As shown in Table 2.1, selecting $d = 2$ minimizes the size of the routing table. Clearly, $d = 2$ is not feasible with previous multilevel clustering protocols, as the size of the hierarchical identifier is unacceptably large. Instead, these protocols are forced to use higher clustering degrees. Regardless of the choice of d, the address size in a regular multilevel clustering protocol is likely to exceed that of a dynamic address routing protocol by at least an order of magnitude, together with a marked increase in routing table size. In the remainder of this section, we assume $d = 2$ for dynamic address routing because this choice of d minimizes the size of the routing table.

Conveniently, with $d = 2$ the *index* can be represented using a single bit. Figure 2.6 shows an example address allocation for a six-node network. Because $\log_2 6 < 3$, 3 bits of address is sufficient for this small network. The most significant bit of the address selects the top-level cluster.

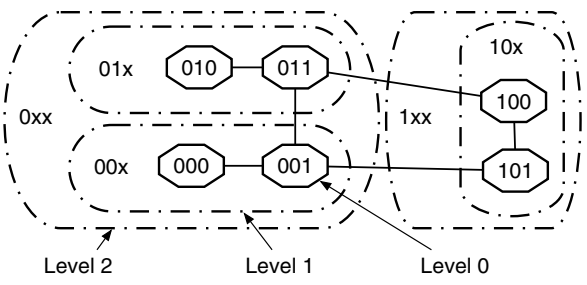


FIGURE 2.6 A network topology and three-level clustering. Nodes have 3-bit routing addresses, with each bit selecting one out of two possible clusters at a given level in the hierarchy.

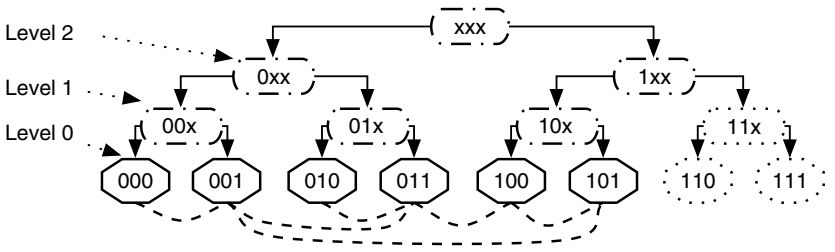


FIGURE 2.7 The address tree of a 3-bit binary address space. Leaves represent actual addresses, whereas inner nodes represent groups of addresses with a common prefix. Dashed lines show physical connectivity between nodes, corresponding to Figure 2.6.

Because $d = 2$, we can also think of the cluster hierarchy as a binary tree, as shown in Figure 2.7. The root of the tree represents the entire network. The leaves of the tree represent nodes and the internal nodes of the tree represent clusters. Each node (leaf) has one routing entry for every level of the tree. This routing entry indicates the path to the other subtree (cluster) at any given level. For example, the node with address [000] would have routing entries for subtrees [001], [01x], and [1xx]. If it wanted to route a packet to the node with address [100], it would look up the routing entry for the subtree [1xx]. After an additional routing step, the packet reaches the node with address [101]. This node has routing entries for subtrees [100], [11x], and [0xx], and is able to forward the packet to its final destination.

One definition of a cluster in the routing context is that the nodes in a cluster form a connected subgraph in the network topology. Because address prefixes uniquely identify clusters in dynamic address routing, nodes with a common address prefix need to have the same property, which is called the *prefix subgraph* constraint. Ensuring that this constraint is satisfied is the primary objective of dynamic address allocation. Next, we describe how this is handled in DART, the Dynamic Address Routing Protocol described by Eriksson et al.[8]

2.9.1 Address Allocation

Dynamic address allocation has many things in common with clustering in multilevel hierarchical networks. However, because dynamic address routing does not rely on concatenating unique identifiers to form its *routing address*, a major concern is to ensure the uniqueness of the addresses allocated.

When a node joins an existing network, it uses the periodic routing updates of its neighbors to identify and select an unoccupied and legitimate address. In more detail, every null entry in a neighbor's routing update indicates an empty subtree. This subtree represents a block of free and valid routing addresses. By definition, the prefix constraint is satisfied if the two subtrees under a given parent are connected, and any empty subtree in a neighbor's routing update by definition shares a parent with the neighbor's subtree at the same level.

Let us see an example of address allocation. Figure 2.8 illustrates the address allocation procedure for a 3-bit address space. Node A starts out alone with address [000]. When node B joins the network, it observes that A has a null routing entry corresponding to the subtree [1xx] and picks the address [100]. Similarly, when C joins the network by connecting to B, C picks the address [110]. Finally, when D joins via A, A's [1xx] routing entry is now occupied. However, the entry corresponding to sibling [01x] is still empty, and so D takes the address [010].

To handle cluster merging and splitting, each cluster, or subtree in this case, is loosely associated with the lowest of all the identifiers of the nodes that belong to that subtree. This is called the *subtree identifier*. With node mobility, subtree identifiers may need to be updated, but these updates are piggybacked on the periodic routing updates at little extra cost. When the node with the lowest identifier within any subtree leaves the subtree, the identifier of that subtree must be recomputed. However, this is generally a nondisruptive process because the route updates from the new lowest identifier node in the subtree will

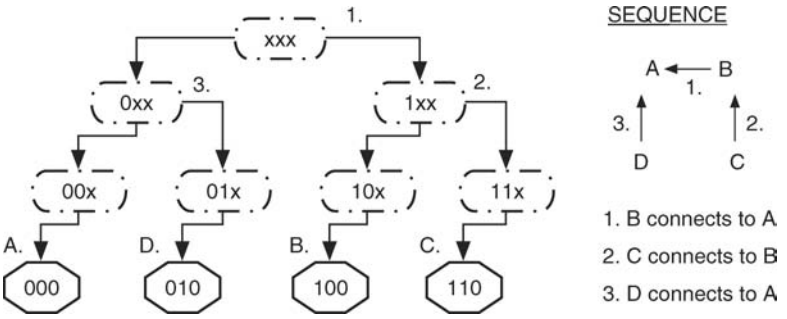


FIGURE 2.8 Address tree for a small network topology. The numbers 1, 2 and 3 show the order in which nodes were added to the network.

propagate and eventually reach all the concerned nodes without forcing any address changes in the process. Note that because of this, the routing address of a node does not depend directly on the identifiers of a set of cluster heads. Therefore, if the node with the lowest identifier gets disconnected, we can expect to see a smaller effect on the cluster hierarchy.

Due to node mobility, clusters will sometimes be partitioned into two or more parts. When this happens, the prefix subgraph constraint does not hold, and the clustering is thus invalid. The solution is to have one of the two partitions acquire new addresses as soon as the partitioning event is detected. The remaining problem is to detect such an event. As described above, subtree identifiers are assigned to be the minimum identifier in the cluster. If the cluster partitions, one of the two partitions will quickly compute a new identifier, as routing updates propagate through the cluster. However, a mere change of the cluster identifier is not enough to accurately diagnose a partitioning event. It could simply be that the node with the lowest identifier went out of range or ran out of battery power. Instead, all route advertisements are made to contain the identifier of the destination subtree. The idea is that in the event that a node receives two routing updates for the same subtree, but with different identifiers, only the update with the lower identifier prevails and gets forwarded further. In addition, when a node perceives a route to its own address subtree, but with a lower identifier, it must acquire a new address. This solution also solves the problem of network merging: if two networks merge, this event will be detected as one or more cluster partitionings, causing some or all of the nodes in one of the two networks to immediately acquire new, valid addresses.

2.9.2 Distributed Location Server

As in several other types of routing protocols, dynamic address routing protocols need a distributed location server. The main problem in designing any distributed location server is to find an effective method to select the *anchor node* of any given identifier. The solution proposed for dynamic address routing protocols is similar to that used in multilevel clustering protocols. However, the methods do not depend on any node identifier, except that of the destination node. A global and *a priori* known function $hash(ID_i)$, which takes a node identifier ID_i and returns a bit string with the same length as the *routing address*, is defined. Second, the $hash(ID_i)$ for the desired destination node i is calculated. If there exists a node that occupies this address, then that node is the *anchor node*. If there is no node with that address, then the node with the most similar address* is the anchor node.

*The metric used here for similarity between addresses can be described as the integer value of the XOR result of the two addresses.

For example, using Figure 2.7 for reference, consider a node with identifier ID_1 that has a current routing address of [010]. This node will periodically send an updated entry to the lookup table, namely $\langle ID_1, 010\rangle$. To figure out where to send the entry, the node uses the hash function to calculate an address: $hash(ID_1)$. If the return value of the hash function is [100], the packet will simply be routed to the node with that address. However, if the returned bit string was instead [111], the packet could not be routed to the node with address [111] because there is no such node. In such a situation, the packet gets routed to the node with the most similar address to [111], which in this case would be [101].

To improve the scalability of the distributed location server, each lookup entry is stored in several locations, at increasing distances from the destination node. By starting with a small, local lookup and gradually going to further away locations, nodes can avoid sending lookup requests across long distances to find a node that is nearby. Similarly, when a node makes a small address change, it need only contact nearby location servers with the location update, as the records at distant location servers will still be sufficiently accurate to guide the packets to the correct neighborhood, where more recent information is readily available.

2.9.2.1 Coping with Temporary Route Failures

On occasion, due to link or node failure, a node will not have a completely accurate routing table. This could potentially lead to lookup packets, both updates and requests, terminating at the wrong node. The end result of this is that requests cannot be promptly served. In an effort to reduce the effect of such intermittent errors, a node can periodically check the lookup entries it stores to see if a route to a more suitable host has been found. If this should be the case, the entry is forwarded in the direction of this more suitable host.

Requests are handled in a similar manner: if the request cannot be responded to with an address, it is kept in a buffer awaiting either the arrival of the requested information, or the appearance of a route to a node that more closely matches the $hash$ of the identifier the request was in regard to. This way, even if a request packet arrives at the anchor node before the update has the anchor, the request will be buffered and served as soon as the update information is available.

Dynamic address routing has size $O(\log N)$ routing tables and an $O(\log N)$ address size. The $\Omega(\log^2 N)$ result for location management handoff shown by Susec and Morsic has not yet been shown for dynamic address routing. However, there are considerable structural similarities between the distributed location server described in that article and the one described for dynamic address routing. Moreover, dynamic address routing has a decreased reliance on node identifiers for clustering and addressing. These observations lead us to conjecture that the lower bound on per-node channel utilization for control packets is, at most, $\Omega(\log^3 N)$.

2.10 Conclusion

This chapter discussed a variety of aspects of ad hoc routing scalability. We deliberated the various routing protocols that have been proposed over the past decade in an effort to understand how these scale with respect to various parameters. To achieve true scalability, it is the belief of the authors that multilevel clustering is the only viable option. While geographic routing is an attractive alternative for certain niche applications, multilevel clustering applies well to all scenarios, except for those with extremely high mobility.

From a scalability perspective, dynamic address routing represents the current state-of-the-art in scalable ad hoc routing. The use of dynamic address routing, a variation on multilevel clustering, results in addresses of length $\Omega(\log N)$. This is considerably shorter than the hierarchical identifiers used in previous multilevel clustering protocols, which are of size $\Omega(\log^2 N)$. Dynamic address routing achieves a similar average routing table size of $\Theta(\log N)$ and offers a reduced dependence on node identifiers for ensuring the stability of clustering and location management.

In summary, scalable ad hoc routing remains a focal point of interest in terms of making the deployment of large-scale ad hoc networks a reality.

References

1. A.D. Amis, R. Prakash, D. Huynh, and T. Vuong. Max-min d-cluster formation in wireless ad hoc networks. In *INFOCOM (1),* pp. 32–41, 2000.
2. S. Basagni, I. Chlamtac, V.R. Syrotiuk, and B.A. Woodward. A distance routing effect algorithm for mobility (DREAM). In *ACM/IEEE MOBICOM,* 1998.
3. B. Chen and R. Morris. L+: scalable Landmark routing and address lookup for multi-hop wireless networks, Tech Report, Massachusetts Institute of Technology, 2002.
4. T. Chen and M. Gerla. Global state routing: a new routing scheme for ad-hoc wireless networks. In *Proc. of IEEE ICC,* 1998.
5. C. Chiang, H. Wu, W. Liu, and M. Gerla. Routing in clustered multihop, mobile wireless networks. In *The IEEE Singapore Int. Conf. on Networks,* 1997.
6. T. Clausen and P. Jaquet. Rfc 3626: Optimized link state routing, 2003.
7. J. Eriksson, M. Faloutsos, and S. Krishnamurthy. Peernet: pushing peer-2-peer down the stack. In *IPTPS:* International Workshop on Peer-to-Peer Systems, 2003.
8. J. Eriksson, M. Faloutsos, and S. Krishnamurthy. Scalable ad hoc routing: the case for dynamic addressing. In *IEEE INFOCOM,* 2004.
9. J.J. Garcia-Luna-Aceves, M. Mosko, and C.E. Perkins. A new approach to on-demand loop-free routing in ad hoc networks. In *Proc. 22nd Annu. Symp. on Principles of Distributed Computing,* pp. 53–62. ACM Press, 2003.
10. M. Grossglauser and D.N.C. Tse. Mobility increases the capacity of ad-hoc wireless networks. In *INFOCOM,* pp. 1360–1369, 2001.
11. P. Gupta and P. Kumar. Capacity of wireless networks, Tech Report, University of Illinois, Urbana-Champaign, 1999.
12. Z. Haas. A new routing protocol for the reconfigurable wireless networks, IEEE International Conference on Universal Personal Communications (ICUPC), 1997.
13. D.B. Johnson and D.A. Maltz. Dynamic source routing in ad hoc wireless networks. In *Mobile Computing,* Vol. 353. Kluwer Academic Publishers, 1996.
14. Brad Karp and H. T. Kung. GPSR: greedy perimeter stateless routing for wireless networks. In *Mobile Computing and Networking,* pages 243–254, 2000.
15. Y.-B. Ko and N.H. Vaidya. Location-aided routing (LAR) in mobile ad hoc networks. In *ACM/IEEE MOBICOM,* 1998.
16. Y.-B. Ko, V. Shankarkumar, and N.H. Vaidya. Medium access control protocols using directional antennas in ad hoc networks. In *INFOCOM (1),* pp. 13–21, 2000.
17. J. Li, J. Jannotti, D. De Couto, D. Karger, and R. Morris. A scalable location service for geographic ad-hoc routing. In *Proc. 6th ACM Int. Conf. on Mobile Computing and Networking (MOBICOM '00)*, pp. 120–130, August 2000.
18. A.K. Mohammed, R.H. Johnson Reidi, David B. Johnson, P. Druschel, and R. Baraniuk. Analysis of safari: an architecture for scalable ad hoc networking and services. Technical Report TREE 0304, Rice University, 2004.
19. S. Murthy and J.J. Garcia-Luna-Aceves. An efficient routing protocol for wireless networks. *Mob. Netw. Appl.,* 1(2):183–197, 1996.
20. M.R. Pearlman and Z.J. Haas. Determining the optimal configuration for the zone routing protocol. *IEEE J. Selected Areas in Communication,* 17(8), August 1999.
21. G. Pei, M. Gerla, and X. Hong. LANMAR: landmark routing for large-scale wireless ad hoc networks with group mobility. In *ACM MobiHOC'00,* 2000.
22. G. Pei, M. Gerla, and T.-W. Chen. Fisheye state routing: a routing scheme for ad hoc wireless networks. In *ICC (1),* pp. 70–74, 2000.
23. G. Pei, M. Gerla, X. Hong, and C.-C. Chiang. A wireless hierarchical routing protocol with group mobility. In *WCNC:* IEEE Wireless Communications and Networking Conference, 1999.

24. C. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *ACM SIGCOMM'94,* 1994.
25. C.E. Perkins and E.M. Royer. Ad-hoc on-demand distance vector routing. In *Proc. Second IEEE Workshop on Mobile Computer Systems and Applications,* pp. 90. IEEE Computer Society, 1999.
26. R. Ramanathan. On the performance of ad hoc networks with beamforming antennas. In *Proc. 2nd ACM Int. Symp. on Mobile Ad Hoc Networking and Computing,* pp. 95–105. ACM Press, 2001.
27. R. Ramanathan and M. Steenstrup. Hierarchically-organized, multihop mobile wireless networks for quality-of-service support. *Mobile Networks and Applications,* 3(1):101–119, 1998.
28. V. Ramasubramanian, Z.J. Haas, and E.G.ün SIRER. SHARP: a hybrid adaptive routing protocol for mobile ad hoc networks. In *Proc. 4th ACM Int. Symp. on Mobile Ad Hoc Networking and Computing,* pp. 303–314. ACM Press, 2003.
29. J. Sucec and I. Marsic. Clustering overhead for hierarchical routing in mobile ad hoc networks. In *IEEE INFOCOM 2002,* New York, June 23–27, 2002.
30. J. Sucec and I. Marsic. Hierarchical routing overhead in mobile ad hoc networks. *IEEE Trans. on Mobile Computing,* 3, Jan. 2004.
31. C.-K. Toh. Associativity-based routing for ad hoc mobile networks. *Wireless Personal Commun. J.,* 4(2):103–139, March 1997.
32. P.F. Tsuchiya. The Landmark hierarchy: a new hierarchy for routing in very large networks. In *SIGCOMM*. ACM, Press, 1988.

3

Uniformly Distributed Algorithm for Virtual Backbone Routing in Ad Hoc Wireless Networks

Dongsoo S. Kim

Ad hoc wireless networks consist of a group of mobile wireless devices. The transmission of a mobile host is received by all hosts within its transmission range due to the broadcast nature of wireless communication and omnidirectional antenna of the mobile hosts. If two wireless hosts are out of their transmission ranges, other mobile hosts residing between them can forward their messages and construct connected networks. Because of the mobility of wireless hosts, each host must be equipped with the capability of an autonomous system, or a router without any statically established infrastructure or centralized administration. In wireless networks, each host can move arbitrarily and can be turned on or off without notifying other hosts. The mobility and autonomy introduces a dynamic topology of the networks not only because end-hosts are transient, but also because intermediate hosts of a communication path are transient.

The ad hoc wireless networks can be modeled using a graph $G = (V, E)$, a vertex $v \in V$ represents a host, and an edge (u, v) indicates that two hosts u and v are within their transmission range. For simplicity, we assume that the communication is bidirectional in that a host u is reachable from a host v iff v is reachable from u, although two or more hosts within a transmission range may not be reachable to each other in the wireless communication due to the hidden-station problem.

The routing problem in ad hoc wireless networks is to find a set of hosts that perform message forwarding, while hosts not included in the set have direct links to at least one host in the set. To construct connected networks, the nodes in the set form a virtual backbone network. A critical issue of such networks is to design efficient routing schemes that can readily adapt to the dynamic topology of the networks. Numerous routing algorithms have been proposed[1,5,7–11,13] recently for ad hoc wireless networks. There are several good surveys on the ad hoc mobile wireless routing protocols.[4,11,12]

3.1 Characteristics of Ad Hoc Networks

Mobility is one of the major characteristics of most ad hoc wireless environments. There are several exceptions where mobility does not need to be considered. In sensor networks, for example, hosts equipped with communication and sensor devices are interconnected to collect data such as temperature or vibration and to report the data to a monitoring center. Most applications of sensor networks require each host to be deployed at a fixed location during its operation. In many other circumstances, however, wireless hosts can move freely in a deployment area, and the connectivity of the hosts is dynamically varying with time. When constructing a virtual backbone, it is necessary to take into account mobility. A global positioning system (GPS) can be used to handle the mobility problem,[15] but its usage must be limited as secondary information because the location information such as latitude, longitude, and height does not directly relate to the connectivity. For example, consider ad hoc networks in transportation trains. Hosts in this application move fast with respect to global position, but their relative locations are practically fixed during travel. In other perspectives, noise in wireless environments and interference by obstacles can make a disconnection between wireless hosts although they are within the normal transmission range. Figure 3.1 shows the relation between motion(M) and connection (C). The shaded area indicates the motion of mobile hosts related to the connection of hosts, in which a global location method can be used to approximate the connectivity of wireless hosts. However, the train example belongs to the category of $M \setminus C$ and the problems of noises and obstacles are in $C \setminus M$, in which GPS-based virtual backbone finding methods are not efficient.

In wireless networks, each host can move arbitrarily (*mobile host's movement*) and can be turned on (*mobile host's switch on*) or off (*mobile host's switch off*) without notifying other hosts.[17] The mobility and autonomy introduce a dynamic topology to the networks not only because end-hosts are transient and mobile, but also because intermediate hosts of a routing path are transient and have the same mobile characteristic. Even in a sensor network where each host tends to stay in a location, moving obstacles such as animals and vehicles introduce the dynamic topology of networks. Every host in ad hoc networks or sensor networks is autonomous, in that their functionalities are identical in the aspect of communication networks (space uniformity). Although it is possible to designate a host as a request initiator or a data collector in some applications, they are generally operating in a distributed environment, and a backbone construction method in ad hoc networks has more constraints than general grid computing. The goal of grid computing is to minimize the computation time to search a final solution utilizing a cluster of computing powers. The physical topology is not altered during its computation and a central node can orchestrate the remaining nodes as needed by an algorithm for finding the solution. In the meantime, a physical topology can be continuously transformed in ad hoc networks due to movement, switch-on, and switch-off. The method of finding a virtual backbone must be able to gradually adapt to the network transformation (time uniformity). This progressive computation plays a critical role in searching virtual backbones in networks. A network state can alter to another before an algorithm finds a backbone unless the transient period is ignorable, meaning the problem set of finding a virtual backbone is time-variant so that its solution must fit into the variation. As a fact of the time-variant, it is desired that the network supports some partial solutions during the transient period. A searching method preferably minimizes the impacts of the transient period. A non-optimal partial solution is acceptable but can provide the network connectivities among mobile hosts before obtaining a final solution.

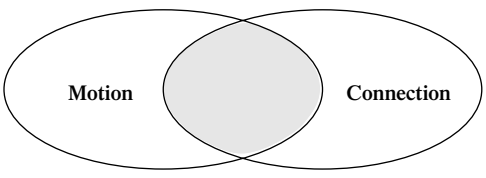


FIGURE 3.1 Set relation of motion and connection.

3.2 Searching Virtual Backbone

For a graph $G = (V, E)$, a subset of the vertex, $V_D \subseteq V$, is a dominating set if each vertex in $V - V_D$ has at least one neighbor in V_D. A virtual backbone network is constructed by connecting the dominating set. To construct the virtual backbone as simple and small as possible, we can use a *minimum connected dominating set* (MCDS). Because finding such an MCDS is an intractable problem,[6] we develop an algorithm for approximating the MCDS that is suitable for use in independent mobile hosts.

For simplicity, we assume that each host is assigned a globally unique identifier and maintains local information, including its dominator, state, a set of neighbors, etc. A host locally broadcasts hello messages in a fixed interval or when its local information is modified so that its neighbors need to update their neighbor data. The local broadcast can be achieved within $O(\Delta)$ under the assumption of a perfect underlying Medium Access Control protocol, where Δ is the maximum degree of the graph.[2] Each host determines its state by comparing its local information to the neighbor information collected through the hello message using a state machine. The machine consists of six states: four permanent states (dominator, essential dominator, dominatee, and absolute dominatee) and two transient states (candidate and dominatee candidate). Each host is initially in the candidate state and moves to one of the permanent states by considering neighbor information.

Many approximation algorithms[3,8,14,16,17] try to find a dominating node set (V_D) in a unit-disk graph $G = (V, E)$ and then add more nodes in V_D using a global leader or a state machine to make the nodes connected. As we see, the resulting CDS is a spanning tree; these approaches are to identify internal nodes, or backbone nodes, first and then to decide leaf nodes, dominated nodes. Our approach, however, uses an opposite direction that recognizes dominated nodes and searches the best dominator for each dominatee by comparing neighbor information. This approach came from a simple observation of geometric graphs, realizing that some nodes do not need to be dominators by directly examining their neighbors, called absolute dominatees. Figure 3.2 illustrates examples of the absolute dominatees. Node 1 has only one neighbor 2, meaning that the node is a leaf in a resulting spanning tree. In other words, edge $(1, 2)$ forms a complete subgraph K_2 and node 1 has no other edge than the subgraph. For the same reason, nodes 3, 4, 5, and 6 are forming a maximum complete subgraph K_4 and nodes 3, 4, and 5 have no additional edge other than K_4. With the maximum complete subgraph, we can designate nodes 3, 4, and 5 as absolute dominatees. Unfortunately, finding a maximum complete subgraph is impractical because it is identical to the NP-complete k-clique problem. For searching an absolute dominatee, however, we can relieve the constraint of the complete subgraph; that is, a node becomes an absolute dominatee if it has a neighbor whose neighbors cover all neighbors. Let N_a denote the neighbors of node a, including the node itself. Node a becomes an absolute dominatee if it has a neighbor b such as $N_a \subset N_b$. For example, node 8 becomes an absolute dominatee because the set of its neighbors $\{7, 8, 9\}$ is included by the neighbor set of node 10, or $\{2, 6, 7, 8, 9, 10\}$. We generated many random connected unit-disk graphs and found out that 40 percent of nodes were absolute dominatees, which validates our approach of quickly identifying dominatees with limited information.

After a node becomes an absolute dominatee, it searches its dominator among its neighbors. A node becomes an essential dominator if its absolute dominatee neighbor considers it as a dominator.

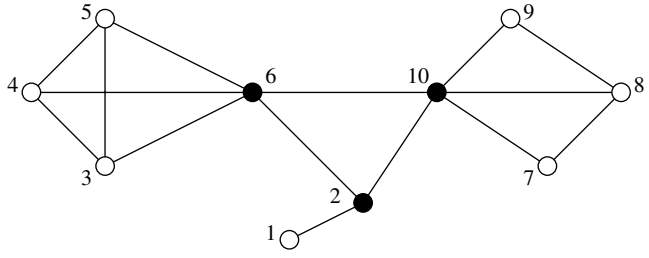


FIGURE 3.2 White nodes are absolute dominatees and black nodes are essential dominators.

For example, nodes 2, 6, and 10 are essential dominators because their neighbors are pointing them as dominators. The absolute dominatees and essential dominators construct trees that will merge together to build a single tree in a connected graph. There are some cases in which an absolute dominatee and an essential dominator cannot be found at all (for example, cyclic graphs or complete graphs), in which a node is a candidate, or the initial state, as all its neighbors are candidates. In this case, the group of nodes start a contention procedure to select a local leader. A simple contention can be achieved using unique identifiers.

Level information and join messages play a critical role in merging subtrees. Each node is initially in level 0, indicating that the node itself is a tree. A node in candidate state looks for its dominator among its neighbors in the dominator state. If the candidate node has a neighbor belonging to a different tree, we can say the node is on a border between two or more subtrees. Each tree has a precedence indicated by its root ID. The precedence of the border node is compared with the precedence of its nearby tree and the node locally broadcasts a join message if it has a higher precedence than the nearby tree. The join message contains the root ID of the nearby tree (old root ID) and the sender's root ID (new root ID). Upon receiving a join message whose old root ID is equal to its own root ID, a node joins to the new tree and searches the best dominator with the new root ID. If the node was not in one of dominatee states, it regenerates and broadcasts a new join message to its neighborhood. For examining the best dominator, the level, the number of children, and the state are taken into consideration.

Figure 3.3 illustrates the algorithm for constructing a virtual backbone. All hosts are initially candidates. After exchanging hello messages at the first round, nodes 0, 1, 6, and 8 identify themselves as absolute dominatees because node 5 (for node 0 and 1), node 2 (for node 6), and node 7 (for node 8) cover the set of neighbors of the corresponding dominatee nodes. Three subtrees are merged together by sending join messages in the next steps. In step 4, node 1 changes its dominator from node 5 to node 2 because nodes 2 and 5 are in the same tree and the level of node 2 in the tree is smaller than the level of node 5.

Figure 3.4 summarizes the state transition of the algorithm. Note that a node state does not move from dominators to dominatees directly, or vice versa. A hello message received by a node does not reflect the network topology at the time when the node calculates its state; rather, the message contains the topology at the previous time interval in the distributed computing environment. The anachronistic information makes it difficult to calculate the dominating set, especially when a node can move, or switch on or off. If the direct transition between dominators and dominatee is allowed, its neighbor can make a premature decision based on timely incorrect information. In the worst case, this miscalculation results in an endless state transition among a set of nodes. To avoid an unstable transition, the candidate states are employed so that a node informs the trends of its state transition to the neighbors.

As explained previously, the topological changes of ad hoc networks are described as three different types. We discuss the recalculation of a connected dominating set caused by topological changes. When a mobile host u switches on, it is in candidate state and all neighbors will not recognize its existence and their existing connectivity will not be affected by the switch-on of node u so that the only possible transition is to be in a dominatee state. The node stays in the dominatee state unless some of its neighbors consider node u as their dominator. When a mobile host is turned off without any notification, its neighbors will not receive a hello message from the node. Each entry in the table for maintaining neighbor information is associated with a timer. If a node does not receive a hello message from a neighbor before the timer expires, it deletes the entry of the neighbor from the table. Eliminating a dominatee node u will not affect the dominatee neighbors; u's dominator might transit to a dominatee if it has no children but node u. If the node turned off is a dominator, its children will look for other nodes as their dominators by deleting the entry for the node after its timer expires. A mobile host movement is viewed as a sequence of connections and disconnections. The event of disconnection can be considered a mobile host switch-off because the event of "out of transmission range" cannot be practically distinguished from the disappearance of the node if we have no geometrical information. The differences between a new connection and switch-on are when the counterpart of a new connection is not in a candidate state. However, when the counterpart v of a node u for a new connection is in a dominatee state, the connection will not affect the computation of u's state because u already has its dominator.

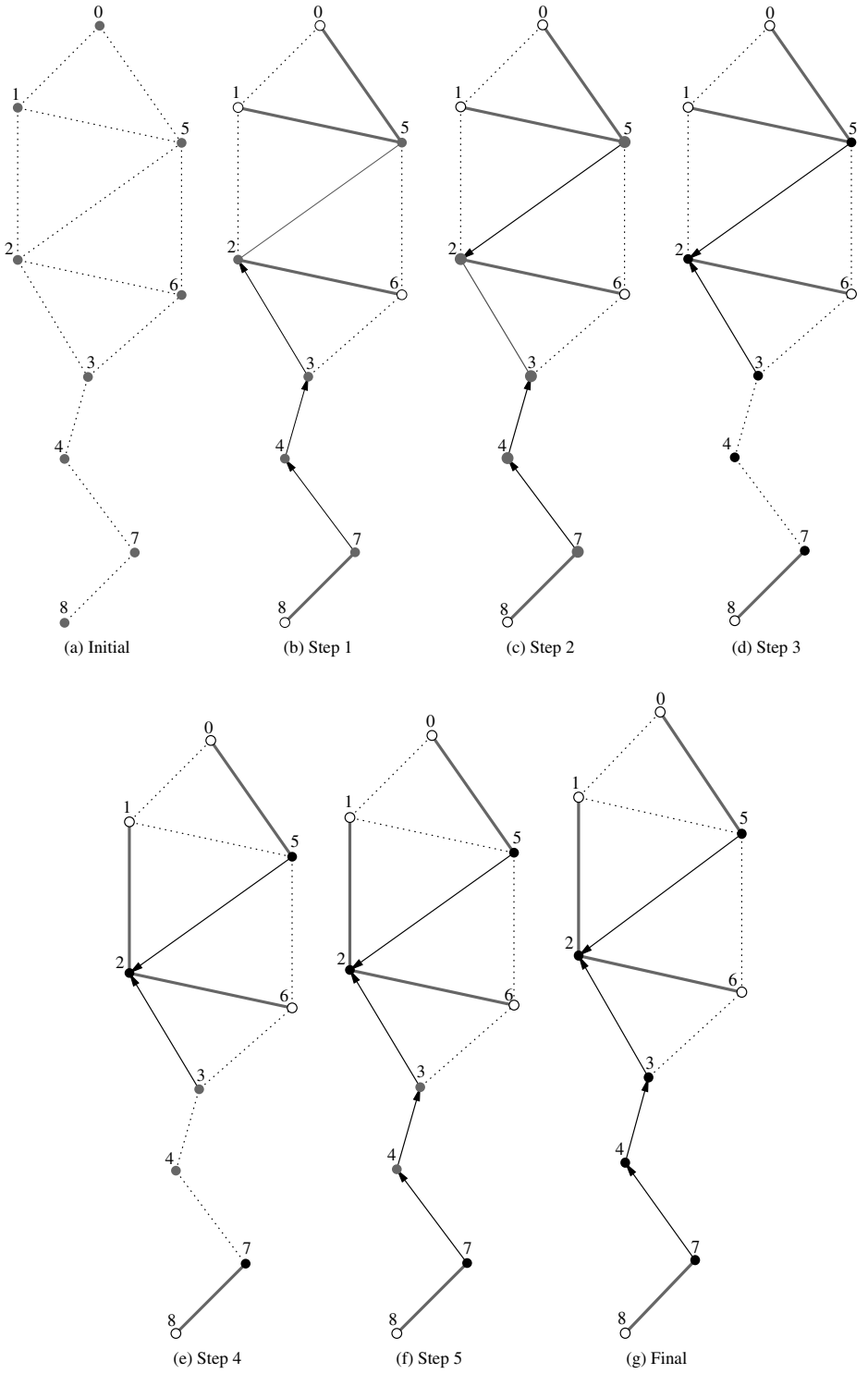


FIGURE 3.3 A sample graph and its state transition. Shaded nodes are in candidate states, hollow nodes are dominatees, and solid nodes are dominators. A dotted line indicates that the edge has no use for building a dominating set. A gray line is an edge between a dominatee and a dominator. A solid line is an edge between two dominators, and their dominating relation is indicated by an arrow.

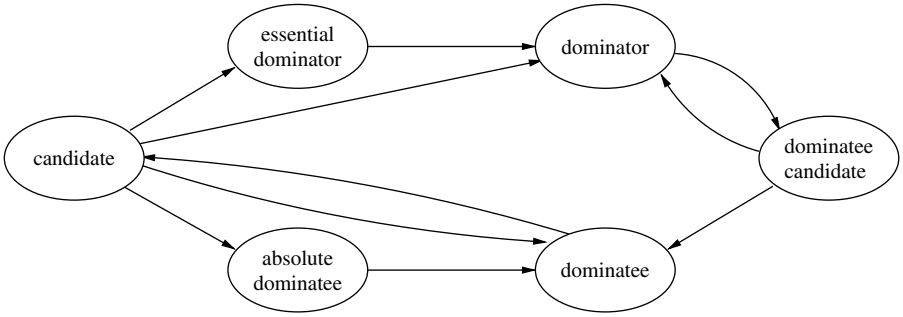


FIGURE 3.4 State transition diagram.

If v is in a dominator state, u will consider v as its dominator when v is beneficial over its previous dominator.

3.3 Conclusion

This chapter proposed a distributed algorithm for constructing a connected dominating set in wireless ad hoc networks. It is distinguished from other approximation algorithms in the aspect of using a bottom-up approach realizing absolute dominatees. The proposed algorithm does not require any geometric information but uses only data that can be obtained from messages exchanged among neighbors. In addition, the algorithm is uniform in terms of space and time so that it is adequate for being deployed in ad hoc networks with no central contol host and no interruption.

References

1. K.M. Alzoubi, P.-J. Wan, and O. Fieder. New distributed algorithm for connected dominating set in wireless ad hoc networks. In *Proc. of HICSS,* 2002.
2. B. Awerbuch. Optimal distributed algorithm for minimum weight spanning tree, counting, leader election and related problems. In *Proc. 19th ACM Symp. on Theory of Computing,* pp. 230–240, 1987.
3. M. Cardei, X. Cheng, and D.Z. Du. Connected domination in multihop ad hoc wireless networks. In *Proc. 6th Int. Conf. on Computer Science and Informatics (CS&I 2002),* North Carolina, March 2002.
4. Y. Chen and A. Liestman. Approximating minimum size weakly-connected dominating sets for clustering mobile ad hoc Networks. In *Proc. of the Symp. on Mobile Ad Hoc Networking and Computing,* 2002.
5. X. Cheng and D.Z. Du. Virtual backbone-based routing in multihop ad hoc wireless networks. In *preparation,* 2002.
6. B.N. Clark, C.J. Colbourn, and D.S. Johnson. Unit disk graph. *Discrete Mathematics,* 86:165–177, 1990.
7. M.S. Corson and A. Ephremides. A distributed routing algorithm for mobile wireless networks. *ACM J. Wireless Networks,* 1(1):61–81, 1995.
8. B. Das and V. Bharghaven. Routing in ad-hoc networks using minimum connected dominating sets. In *Proc. Int. Conf. on Communication (ICC97),* pp. 376–380, 1997.
9. D.B.Johnson. Routing in ad hoc networks of mobile hosts. In *Proc. of Workshop on Mobile Computing Systems and Applications,* pp. 158–163, Dec. 1994.
10. S. Guha and S. Khuller. Approximation algorithms for connected dominating sets. *Algorithmica,* 20(4):374–387, 1998.
11. P. Krishna, M. Chatterjee, N.H. Vaidya, and D.K. Pradhan. A cluster-based approach for routing in ad-hoc networks. In *Proc. Second USENIX Symposium on Mobile and Location-Independent Computing,* pp. 1–10, 1995.

12. E.M. Royer and C.-K. Toh. A review of current routing protocols for ad-hoc mobile wireless networks. *IEEE Personal Communications,* pp. 46–55, 1999.
13. R. Sivakumar, B. Das, and V. Bharghavan. An improved spine-based infrastructure for routing in ad hoc networks. In *Proc. Int. Symp. on Computers and Communications (ISCC'98),* 1998.
14. I. Stojmenovic, M. Seddigh, and J. Zunic. Dominating sets and neighbor elimination-based broadcasting algorithm in wireless networks. In *Proc. IEEE Hawaii International, Conference on System Science,* January 2001.
15. I. Stojmenovic, M. Seddigh, and J. Zunic. Dominating sets and neighbor elimination-based broadcasting algorithms in wireless networks. *IEEE Trans. on Parallel and Distributed Systems,* 12(12), December 2001.
16. P.-J. Wan, K.M. Alzoubi, and O. Frieder. Distributed construction of connected dominating set in wireless ad hoc networks. In *Proc. on the Joint Conf. of the IEEE Computer and Communication Societies (INFOCOM),* pp. 1597–1604, March 2002.
17. J. Wu and H. Li. On calculating connected dominating set for efficient routing in ad hoc wireless networks. In *Proc. 3rd Int. Workshop on Discrete Algorithms and Methods for Mobile Computing and Communication,* pp. 7–14, Seattle, WA, 1999.

4

Maximum Necessary Hop Count for Packet Routing in MANETs

Xiao Chen

Jian Shen

This chapter investigates a fundamental characteristic of a mobile ad hoc network (MANET): the maximum necessary number of hops needed to deliver a packet from a source to a destination. In this chapter, without loss of generality, we assume that the area is a circle with a radius of r, $r > 1$, and the transmission range of each mobile station is 1. We prove that the maximum necessary number of hops needed to deliver a packet from a source to a destination is $\frac{4\pi}{\sqrt{3}}(r + \frac{1}{\sqrt{3}})^2 - 1 = \frac{4\pi r^2}{\sqrt{3}} + O(r) \approx 7.255r^2 + O(r)$. We show that this result is very close to optimum with only a difference of $O(r)$.

4.1 Introduction

Recent advances in technology have provided portable computers with wireless interfaces that allow networked communication among mobile users. The resulting environment no longer requires users to maintain a fixed and universally known position in the network and enables almost unrestricted mobility.

A *mobile ad hoc network (MANET)* is formed by a cluster of mobile stations randomly located within a certain area without the infrastructure of base stations. The applications of MANETs appear in places where predeployment of network infrastructure is difficult or unavailable (e.g., fleets in oceans, armies in march, natural disasters, battlefield, festival field grounds, and historic sites).

In a MANET, stations communicate with each other by sending and receiving packets. The delivery of a packet from a source station to a destination station is called *routing*. Particularly in a MANET, two stations can communicate directly with each other if and only if they are within each other's *wireless transmission range*. Otherwise, the communication between them must rely on other stations. For example, in the

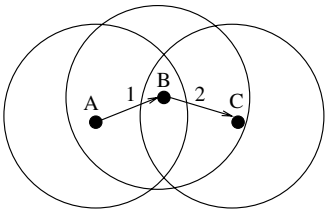


FIGURE 4.1 Example ad hoc wireless network.

network shown in Figure 4.1, stations A and B are within each other's transmission range (indicated by the circles around A and B, respectively). If A wants to send a packet m to B, A can send it directly in one hop. A and C are not within each other's transmission range. If A wants to send a packet to C, it must first forward the packet to B and then use B to route the packet to C. Therefore, it takes two hops to deliver a packet from A to C.

Many routing algorithms have been designed[1,3,5,6,8] but less work has been done to investigate the fundamental properties of MANETs in a mathematical way. This chapter addresses the issue of finding the maximum necessary number of hops needed to deliver a packet from a source to a destination in a MANET within a certain area. An initial attempt by us to solve the problem was made.[2] In this chapter we reconsider the problem and provide a much better maximum necessary hop count. Without loss of generality, we assume that all the mobile stations are located within an area of a circle with a radius r, $r > 1$, and the transmission range of all the nodes is 1, assuming they use the fixed transmission power.

This chapter is organized as follows. Section 4.2 provides the notations. Section 4.3 puts forward the problem. Section 4.4 introduces the circle packing problem, and Section 4.5 provides our solution to the problem. Section 4.6 shows the sharpness of our solution, and Section 4.7 is the conclusion.

4.2 Notations

We can use a simple graph $G = G(V, E)$ to represent a MANET, where the vertex set V is the collection of mobile stations within the wireless network. An edge between two stations u and v denoted by $u \leftrightarrow v$ means that both of them are within each other's transmission range. We assume that this graph G is finite and connected.

See an example of a wireless network in Figure 4.2. There are five stations A, B, C, D, and E in a MANET. The circle around each one represents its transmission range. Two vertices are connected if and only if they are within each other's transmission range. The resultant graph is shown in Figure 4.3.

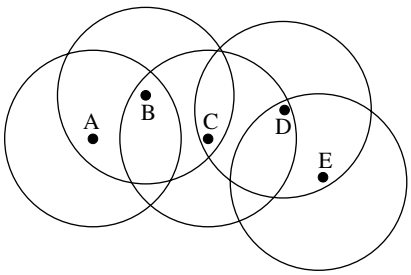


FIGURE 4.2 Example ad hoc wireless network.

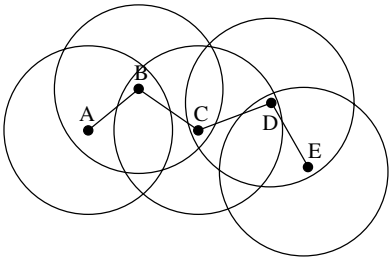


FIGURE 4.3 The graph representing an ad hoc wireless network.

4.3 The Problem

This chapter addresses the issue of finding the maximum necessary number of hops needed to deliver a packet from a source to a destination in a MANET within a certain area. Without loss of generality, we assume that all the mobile stations are within an area of a circle with a radius r, $r > 1$. Two stations u and v can communicate with each other if and only if their geographic distance is less than or equal to 1. Based on the above description, a simple graph G can be drawn to represent the MANET within this circle. The maximum necessary number of hops to deliver a packet from a source to a destination is actually the diameter of the graph G.

4.4 Circle Packing Problem

Before presenting our result, we introduce the circle packing problem that leads to our solution.

The *circle/sphere packing* problem is to consider how to effectively pack non-overlapping small circles/spheres of the same size into a large circle/sphere as many as possible so that the *density of a packing*, which is the ratio of the region/space occupied by the circles/spheres to the whole region/space, is as large as possible.

The history of the circle/sphere packing problem goes back to the early 1600s when astronomer-mathematician Johannes Kepler asserted that no sphere packing could be better than face-centered cubic (FFC) packing. FFC packing is the natural one that arises from packing spheres in a pyramid, as shown in Figure 4.4.

The Kepler Conjecture. *The density of any sphere packing in three-dimensional space is at most* $\pi/\sqrt{18}$*, which is the density of the FCC packing.*

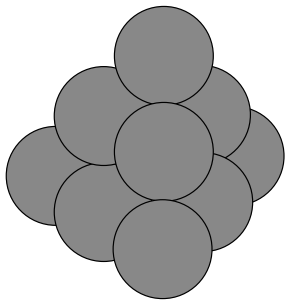


FIGURE 4.4 A pyramid in face-centered cubic sphere packing.

Although the Kepler conjecture looks natural, it is very difficult to prove. Recently, a 250-page proof of the Kepler conjecture was claimed.[4]

Compared with the difficulties of the Kepler conjecture, the problem of circle packing in the plane has been solved with ease. The result is stated in the following lemma. It was first proved by Axel Thue in 1890. To make this chapter self-contained, we include a proof provided by Surendran[7] with the following slight change: while the proof in Ref. 7 uses unit circles to pack the plane, we use circles of radius 1/2 to pack the plane in order to be consistent with the proof of Theorem 4.1 in Section 4.5.

Lemma 4.1 *The optimum packing of circles in the plane has density* $\pi/\sqrt{12}$*, which is that of the hexagonal packing shown in Figure 4.5.*

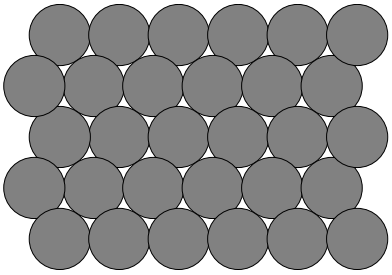


FIGURE 4.5 The hexagonal circle packing of the plane.

Proof We can suppose without loss of generality that we use circles of radius $1/2$ to pack the plane. Start with an arbitrary packing of circles of radius $1/2$ in the plane. Around each circle draw a concentric circle of radius $1/\sqrt{3}$. (The reason for choosing $1/\sqrt{3}$ as the radius of the concentric circle is that when three $1/2$ radius circles are packed next to each other just as shown in the left part of Figure 4.6, the three centers can form an equilateral triangle, then the three cooresponding concentric circles can intersect at the middle of the equilateral triangle.) Where a pair of concentric circles overlap, join their points of intersection and use this as the base of two isosceles triangles. The centers of each circle form the third vertex of each triangle. In this way, the plane can be partitioned into three regions, as depicted in Figure 4.6.

1. The isosceles triangles: if the top angle of the triangle is θ radians, then its area is $(1/\sqrt{3})^2 \sin\theta/2 = \sin\theta/6$ and the area of the sector is $(1/2)^2\theta/2 = \theta/8$, so that the packing density is

$$\frac{\theta/8}{\sin\theta/6} = \frac{3\theta}{4\sin\theta},$$

   where θ ranges between 0 and $\pi/3$ radians. The maximum value of the density is $\pi/\sqrt{12}$, attained at $\theta = \pi/3$ radians.
2. Regions of the larger circles not in a triangle: here the regions are sectors of a pair of concentric circles of radius $1/2$ and $1/\sqrt{3}$, so the density is

$$\left(\frac{1/2}{1/\sqrt{3}}\right)^2 = \frac{3}{4} < \frac{\pi}{\sqrt{12}}$$

3. Regions not in any circle: here the density is zero.

Because the density of each region of space is at most $\pi/\sqrt{12}$, the density of this circle packing is at most $\pi/\sqrt{12}$. Furthermore, a packing can only be optimum when it causes space to be divided into equilateral triangles. This only happens for the hexagonal packing of Figure 4.5. □

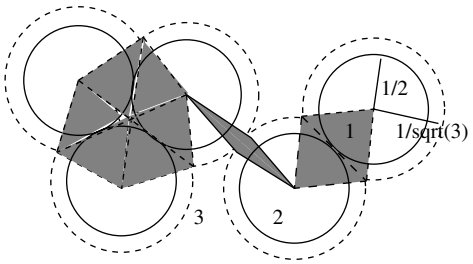


FIGURE 4.6 The plane partition induced by a sample circle packing.

Remarks The above lemma concerns circle packing into an unlimited space. Now consider that some circles C_i of radius 1/2 are packed into a limited region such as a large circle with a radius t. From now on, we use the notation $C(x)$ to represent a circle with a radius x. In the proof of Lemma 4.1, we draw a concentric circle $C(1/\sqrt{3})$ around each C_i. Because all circles C_i are within a circle $C(t)$, all those circles $C(1/\sqrt{3})$ must be within the circle $C(t + 1/\sqrt{3} - 1/2)$, which is concentric with $C(t)$. Thus, the proof of Lemma 4.1 can still be applied to the limited region $C(t + 1/\sqrt{3} - 1/2)$; that is,

$$\frac{\sum_i (\text{area of } C_i)}{\text{area of } C(t + 1/\sqrt{3} - 1/2)} \leq \frac{\pi}{\sqrt{12}}$$

Furthermore, Lemma 4.1 shows that $\pi/\sqrt{12}$ is the best upper bound for the ratio of the area of all C_i's to the area of $C(t + 1/\sqrt{3} - 1/2)$, in the case that $C(t)$ is sufficiently larger than each of the small circles C_i.

4.5 Our Solution

Based on the above lemma, the following is our result of the maximum necessary hop count to deliver a packet from a source to a destination in MANET.

Theorem 4.1 *Assume that all the mobile stations are within an area of a circle with a radius r, $r > 1$. The transmission range of each mobile station is* 1. *Denote the graph generated by connecting all pairs of vertices within each other's transmission range as G; that is, two vertices are connected if and only if their geographic distance is less than or equal to* 1. *Then an upper bound for the diameter of G is $\frac{4\pi}{\sqrt{3}}(r + \frac{1}{\sqrt{3}})^2 - 1$; in other words, it takes maximum $\frac{4\pi}{\sqrt{3}}(r + \frac{1}{\sqrt{3}})^2 - 1 = \frac{4\pi r^2}{\sqrt{3}} + O(r)$ necessary hops to deliver a packet from a source to a destination.*

Proof Let D be the diameter of the graph G. Choose two vertices u, v such that the distance in G between u and v, $d_G(u, v)$, is D; that is, $d_G(u, v) = D$. Then there exist distinct vertices u_i, $1 \leq i \leq D+1$, such that

$$u = u_1 \leftrightarrow u_2 \leftrightarrow u_3 \leftrightarrow \cdots \leftrightarrow u_D \leftrightarrow u_{D+1} = v$$

is a shortest path of length D between u and v (see Figure 4.7).

We define a set $I = \{u_i : i \text{ is odd}\}$. Then the size of I, denoted by $|I|$, is $\lceil (D+1)/2 \rceil$. We can prove that I is an *independent set* of vertices. An independent set of vertices is defined as a set of vertices in which there is no edge between any pair of vertices in the set. □

Claim 4.1 I is an independent set of vertices in graph G.

Proof of Claim 4.1 Suppose that I is not an independent set; that is, there exists at least one edge between some pair of vertices in I. Without loss of generality, we assume that there are two vertices u_{2j+1}, u_{2k+1} in I such that $j < k$ and $u_{2j+1} \leftrightarrow u_{2k+1}$. By the definition of I, the two vertices are on the shortest path from u to v. Then the shortest path can be represented as

$$u = u_1 \leftrightarrow \cdots \leftrightarrow u_{2j} \leftrightarrow u_{2j+1} \leftrightarrow u_{2k+1}$$
$$\leftrightarrow u_{2k+2} \leftrightarrow \cdots \leftrightarrow u_{D+1} = v$$

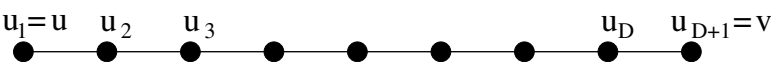


FIGURE 4.7 A shortest path between u and v.

The length of this path, as represented, is $D - [(2k+1) - (2j+1)] + 1 = D + 2j - 2k + 1$. It is less than D. This contradicts to $d_G(u, v) = D$. Therefore, there is no edge between any pair of vertices in I; in other words, I is an independent set in G.

By Claim 4.1 and the definition of G, we have the following claim.

Claim 4.2 The geographic distance between any pair of vertices in I is larger than 1.

Then, for each vertex $u_i \in I$, we define a circle S_i with a center at u_i and with a radius of 1/2. By Claim 4.2, two circles S_j, S_k are disjoint if $j \neq k$. Because all the vertices in I are covered by the circle $C(r)$, all the disjoint circles S_i (i is odd) can be covered by a larger circle named $C(r + 1/2)$, which has the same center as the circle $C(r)$ and has a radius of $r + 1/2$ (see Figure 4.8 for an example). Now we can relate this diameter problem to the circle packing problem, that is, how to effectively pack these non-overlapping circles S_i (i is odd) as many as possible into the larger circle $C(r + 1/2)$. By the remarks following the proof of Lemma 4.1 in Section 4.4 (note that the circle $C(t + 1/\sqrt{3} - 1/2)$ in the remark should be converted to $C(r + 1/\sqrt{3})$ since $t = r + 1/2$.), we have

$$\frac{\sum_i (\text{Area of } S_i)}{\text{Area of } C(r + 1/\sqrt{3})} \leq \frac{\pi}{\sqrt{12}}$$

where $C(r + 1/\sqrt{3})$ is the circle that has the same center as the circle $C(r)$ and has a radius of $r + 1/\sqrt{3}$. Thus,

$$\frac{|I|\pi \left(\frac{1}{2}\right)^2}{\pi \left(r + \frac{1}{\sqrt{3}}\right)^2} \leq \frac{\pi}{\sqrt{12}}$$

Solving for $|I|$,

$$|I| \leq \frac{2\pi}{\sqrt{3}} \left(r + \frac{1}{\sqrt{3}}\right)^2$$

Since $|I| = \lceil (D+1)/2 \rceil \geq (D+1)/2$, we have

$$D \leq 2|I| - 1 \leq \frac{4\pi}{\sqrt{3}} \left(r + \frac{1}{\sqrt{3}}\right)^2 - 1$$

So, it takes maximum $\frac{4\pi}{\sqrt{3}}(r + \frac{1}{\sqrt{3}})^2 - 1 = \frac{4\pi r^2}{\sqrt{3}} + O(r)$ necessary hops to deliver a packet from a source to a destination.

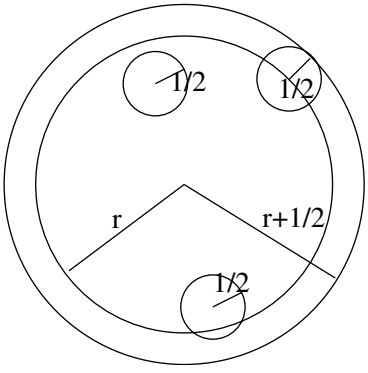


FIGURE 4.8 All the S_i's are covered by a circle of radius $r + 1/2$.

4.6 Sharpness of the Maximum Necessary Hop Count

In this section, we show the sharpness of the maximum necessary hop count. The idea is like this: if we can actually construct a graph G with a diameter of $4\pi r^2/\sqrt{3} + O(r)$ in a circle $C(r)$, then our maximum necessary hop count $\frac{4\pi}{\sqrt{3}}(r + \frac{1}{\sqrt{3}})^2 - 1$ is very close to optimum, with only a difference of $O(r)$. The construction of such a G is based on a parallelogram packing into the circle $C(r)$. Before the construction, we present some assumptions and properties of the packing unit "parallelogram."

Let ϵ $(0 < \epsilon < 1)$ be a positive real number. We draw a parallelogram $ABDE$. Then we choose points C and G on the side of BD, and choose a point F on the side of AE such that $|AC| = |AG| = 1+\epsilon$, $|AF| = 1$, $|CG| = 1 - \epsilon$, and $|BG| = |CD| = |EF| = \epsilon$, where $|\ |$ represents the length of a line segment. (See Figure 4.9.) Then $|AB|$ and $|DE|$ can be uniquely determined in terms of ϵ because $ABDE$ is a parallelogram. Next we give some properties of the parallelogram.

Property 4.0 $|AF| = |BC| = 1$, $|EF| = |CD| = \epsilon$, and $|AE| = |BD| = 1 + \epsilon$.

This property is obvious from the above assumptions.

Property 4.1 Any two vertices chosen from sets $\{A, E, F\}$ and $\{B, C, D\}$, respectively, are at least $1 + \epsilon$ distance apart.

Proof of Property 4.1 First, since $|AC| = |AG|$ and $|CD| = |GB|$, elementary geometry shows that the triangles ACD and AGB are identical. Thus, $|AD| = |AB|$, angle $\angle ADC$ is an acute angle, and angle $\angle ACD$ is an obtuse angle. Then, $\angle ACD > \angle ADC$ implies $|AD| > |AC|$, so

$$|AB| = |AD| > |AC| = 1 + \epsilon$$

Second, because $|AF| = |GD| = 1$, we know that $AGDF$ is a parallelogram. This implies $|FD| = |AG|$ and $\angle FDB = \angle AGB = \angle ACD > \pi/2$ radians. Thus,

$$|FB| > |FC| > |FD| = |AG| = 1 + \epsilon$$

Third, because $\angle EDB > \angle FDB > \pi/2$ radians, we have

$$|EB| > |EC| > |ED| = |AB| > |AC| = 1 + \epsilon.$$

Therefore, any two vertices chosen from sets $\{A, E, F\}$ and $\{B, C, D\}$, respectively, are at least $1 + \epsilon$ distance apart.

Property 4.2 The area of the parallelogram $ABDE$ is $\frac{1+\epsilon}{2}\sqrt{3 + 10\epsilon + 3\epsilon^2}$.

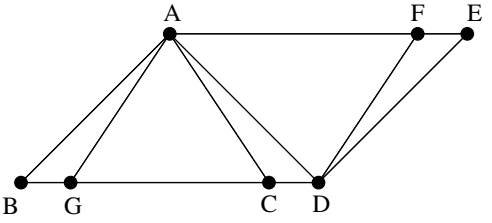


FIGURE 4.9 A parallelogram ABDE.

Proof of Property 4.2 By the Pythagoras theorem, the height of the parallelogram $ABDE$ is $\sqrt{|AC|^2 - (|CG|/2)^2}$. Thus, the area of the parallelogram is

$$|BD| \cdot \sqrt{|AC|^2 - \left(\frac{|CG|}{2}\right)^2} = (1+\epsilon) \cdot \sqrt{(1+\epsilon)^2 - \left(\frac{1-\epsilon}{2}\right)^2}$$
$$= \frac{1+\epsilon}{2}\sqrt{3+10\epsilon+3\epsilon^2}$$

Property 4.3 $|BE| = \sqrt{3+7\epsilon+3\epsilon^2}$.

Proof of Property 4.3 Draw a line segment EH perpendicular to the line BD, as shown in Figure 4.10. Then $|EH|$ is the height of the parallelogram, and so

$$|EH| = \sqrt{|AC|^2 - \left(\frac{|CG|}{2}\right)^2} = \frac{1}{2}\sqrt{3+10\epsilon+3\epsilon^2}$$

Applying the Pythagoras theorem to the right triangle BHE,

$$|BE| = \sqrt{|EH|^2 + |BH|^2} = \sqrt{|EH|^2 + \left(|AE| + \frac{|BD|}{2}\right)^2}$$
$$= \sqrt{\frac{3+10\epsilon+3\epsilon^2}{4} + \left(\frac{3(1+\epsilon)}{2}\right)^2} = \sqrt{3+7\epsilon+3\epsilon^2}$$

Now let P denote the parallelogram $ABDE$ with six points A, B, C, D, E, F on its boundary. This is our packing unit parallelogram. We use parallel packing to pack as many non-overlapping P's as possible into the circle $C(r)$, as shown in Figure 4.11. Let

$$l = \sqrt{3+7\epsilon+3\epsilon^2}$$

By Property 4.3, any two points in P are at most l distance apart. Then the circle $C(r-l)$ concentric with $C(r)$ is entirely covered by P's because otherwise we could have packed one more P without reaching the boundary of $C(r)$. Thus, by Property 4.2, at least n parallelograms P can be packed into $C(r)$, where

$$n \geq \frac{\text{Area of } C(r-l)}{\text{Area of } P} = \frac{\pi(r-l)^2}{(1+\epsilon)/2\sqrt{3+10\epsilon+3\epsilon^2}} = \frac{2\pi(r-\sqrt{3+7\epsilon+3\epsilon^2})^2}{(1+\epsilon)\sqrt{3+10\epsilon+3\epsilon^2}}$$

Let V be the set of all points A, B, C, D, E, F in P's; that is, $V = \cup_P \{A, B, C, D, E, F\}$ and let $|V|$ denote the number of points in V. We have the following property:

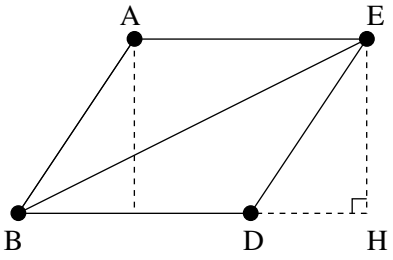


FIGURE 4.10 Computation of $|BE|$.

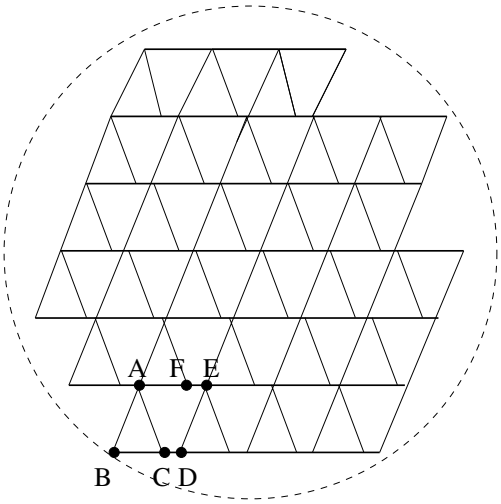


FIGURE 4.11 Parallel packing of parallelograms into a circle.

Property 4.4 $|V| \geq 2n$.

Proof of Property 4.4 We can assume that the parallelograms are packed one by one into $C(r)$, from the lower layer to the higher layer, and in each layer from the left to the right. Because each time adding a parallelogram P increases $|V|$ by at least 2, Property 4.4 holds by induction.

Having the above assumptions and properties of parallelogram, now we construct a graph G within $C(r)$ as follows: let V be the vertex set of G and two vertices in V are connected if and only if they are at most 1 distance apart. By Property 0, vertices in each layer are connected to form a path; and by Property 4.1, any two vertices in different layers are not connected. So G is a union of disjoint paths. We can add some new vertices to G and let them be the intermediate vertices connecting paths of two adjacent layers, as shown in Figure 4.12. Because at least 1 new vertex is added to G, the final graph is a path with at least

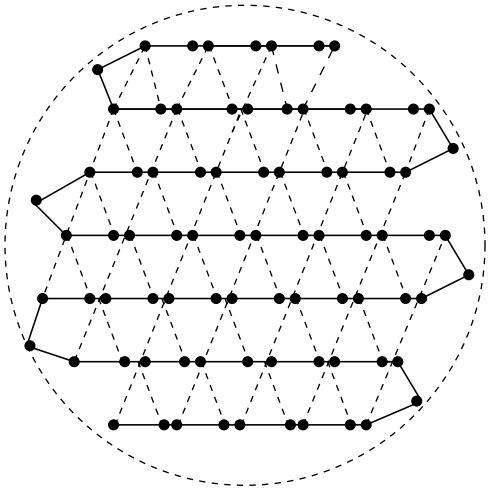


FIGURE 4.12 The construction of a path.

$|V| + 1$ vertices. So it has a diameter of at least

$$|V| \geq 2n \geq \frac{4\pi(r - \sqrt{3 + 7\epsilon + 3\epsilon^2})^2}{(1+\epsilon)\sqrt{3 + 10\epsilon + 3\epsilon^2}}$$

Because the above argument holds for all the real numbers ϵ with $0 < \epsilon < 1$, we can make ϵ approaching 0. Because

$$\lim_{\epsilon \to 0} \frac{4\pi(r - \sqrt{3 + 7\epsilon + 3\epsilon^2})^2}{(1+\epsilon)\sqrt{3 + 10\epsilon + 3\epsilon^2}} = \frac{4\pi}{\sqrt{3}}(r - \sqrt{3})^2 = \frac{4\pi r^2}{\sqrt{3}} + O(r)$$

a graph with a diameter of $4\pi r^2/\sqrt{3} + O(r)$ always exists (by choosing ϵ very close to 0). Therefore, our upper bound for the diameter (maximum necessary hop count) in Theorem 4.1,

$$\frac{4\pi}{\sqrt{3}}\left(r + \frac{1}{\sqrt{3}}\right)^2 - 1 = \frac{4\pi r^2}{\sqrt{3}} + O(r)$$

is very close to optimum with only a difference of $O(r)$.

4.7 Conclusion

The maximum necessary number of hops needed to deliver a packet from a source to a destination has been found for a MANET within a circle of radius r, $r > 1$, assuming the transmission range of each mobile station is 1. Our proofs show that our result is very close to optimum, with only a difference of $O(r)$.

References

1. V. Bharghavan, A. Demers, S. Shenker, and L. Zhang, MACAW: A Medium Access Protocol for Wireless LANs, *Proc. of SIGCOMM'94,* 1994.
2. X. Chen, J. Shen, and X.D. Jia, An Upper Bound for a Mobile Ad-Hoc Wireless Network, *Proc. of the Int. Conf. on Parallel and Distributed Processing Techniques and Applications,* June 2001, pp. 1617–1620.
3. C.L. Fullmer and J.J. Garcia-Luna-Aceves, Floor Acquisition Multiple Access (FAMA) for Packet-Radio Networks, *Proc. of SIGCOMM'95,* 1995.
4. Web page of Prof. T. Hales, http://www.math.pitt.edu/∼thales/
5. C.R. Lin and M. Gerla, MACA/PR: An Asynchronous Multimedia Multihop Wireless Network, *Proc. of INFOCOM'97,* 1997.
6. C.R. Lin and M. Gerla, Real-Time Support in Multihop Wireless Network, *ACM/Baltzer Wireless Networks,* 5(2), 1999.
7. D. Surendran, The Conquest of the Kepler Conjecture, *Math Horizons,* 8, 8–12, (2001).
8. J. Wu and H. Li, A dominating-Set-Based Routing Scheme in Ad Hoc Wireless Networks, special issue on Wireless Networks in the *Telecommunication Systems Journal,* 3, 63–84, 2001.

5

Efficient Strategy-Proof Multicast in Selfish Wireless Networks

Xiang-Yang Li

Weizhao Wang

In this chapter, we study how to perform routing when each wireless node is selfish; that is, a wireless node will always maximize its own benefit. Traditionally, it is assumed by the majority of the routing protocols for wireless networks that each wireless node will forward the packets for other nodes if it is asked to do so. However, this assumption may not be true in practice, especially when the wireless devices are owned by individual users. A node will deviate from a routing protocol if it will gain more benefit by doing so. In this chapter, we assume that each wireless node will incur a cost when it forwards a unit

of data for some other nodes. A node will forward the data only if it gets a payment to compensate its cost. Its profit (or called utility) will then be the payment minus its cost if it did forward the data. For a multicast with a source node and a set of receiver nodes, we assume that they will pay the relay nodes to carry the traffic from the source to receivers. We assume that the cost of each agent is private and each agent can manipulate its reported cost to maximize its utility. A payment scheme is *strategyproof* if every agent maximizes its utility when it reports its cost truthfully. In this chapter, we propose several strategyproof mechanisms for multicast in selfish wireless networks when each node has a cost of forwarding a unit data based on various structures. We prove that each of our payment schemes is optimum for the corresponding structure used.

5.1 Introduction

Recent years saw a great amount of research into wireless ad hoc networks on various important problems such as routing, quality of service, security, power management, and traffic and mobility modeling. However, there are still many challenges left. In wireless ad hoc networks, each host contributes its local resources to forward the data for other nodes to serve the common good, and may benefit from resources contributed by other hosts in return. Based on such a fundamental design philosophy, wireless networks provide appealing features of enhanced system robustness, high service availability, and scalability. However, the critical observation that users are generally selfish and noncooperative may severely undermine the expected wireless structure. For example, for a routing algorithm based on the least cost path (LCP), the individual wireless node may declare an arbitrarily high cost for forwarding a data packet to other nodes because wireless nodes are energy-constrainted and it is often not in the interest of a node to always relay the messages for other nodes. The root cause of the problem is, obviously, that there exist no incentives for users to be altruistic. Following the common belief in neoclassic economics, it is more reasonable to assume that all wireless terminals are *rational*: they try to maximize their benefits instead of conforming to the existing protocols. Thus, we need to design some mechanisms to ensure that these *rational* wireless terminals will conform to our protocols without any deviation.

How to achieve cooperation among wireless terminals in network was previously addressed.[2–4,11,13,16,17] The key idea behind these approaches is that terminals providing a service should be remunerated, while terminals receiving a service should be charged. Both of these methods belong to the so-called *credit-based method*. Some of these algorithms need some special hardware that is not very practical in the real world. In recent years, *incentive-based methods* have been proposed to solve the noncooperative problem. The most well-known and widely used incentive-based methods is the so-called family of VCG mechanisms by Vickrey,[18] Clarke,[5] and Groves.[9] Nisan and Ronen[14] provided a mechanism belonging to the VCG family to assure the cooperation for the unicast problem in a general network where each communication link is assumed to be selfish and rational.

While unicast in wireless network has been studied extensively in the literature and deployed in practice for years, several important issues about multicast over wireless networks have not been explored fully. In practice, multicast is a more efficient way to support group communication than unicast or broadcast, as it can transmit packets to destinations using fewer network resources, which is critical in wireless networks. Typical wireless multicast applications include group-oriented mobile commerce, military command and control, distance education, and intelligent transportation systems. For a multicast routing, usually a tree with the minimum cost that spans the sources and receivers is used because it requires less network resources than other structures. Finding such a minimum cost tree is known to be NP-hard. Thus, some multicast trees with good practical performances have been proposed in the literature. Unlike unicast problem, as we will show later, if we simply apply a VCG mechanism to those commonly used multicast tree structures, we cannot guarantee that all wireless devices will follow our prescribed protocols. In this chapter, we discuss how to design truthful non-VCG mechanisms for those multicast structures in *selfish* wireless networks.

The rest of the chapter is organized as follows. In Section 5.2, we review some definitions and *priori* art on truthful mechanism design for multicast. In Section 5.3, we present the first strategyproof mechanism

for the Steiner tree problem (or multicast). The output of our mechanism (a tree) has a cost within a constant factor of the optimum, and the payment is minimum among any truthful mechanism having this output. In Section 5.4, we show our experimental study of our proposed mechanisms. We conclude the chapter in Section 5.5 by pointing out some possible future work.

5.2 Preliminaries and Priori Art

5.2.1 Wireless Ad Hoc Networks

Wireless *ad hoc* networks are emerging as a flexible and powerful wireless architecture that does not rely on a fixed networking infrastructure. Wireless ad hoc networks have received significant attention over the past few years due to their potential applications in various situations such as battlefield, emergency relief and environmental monitoring, etc. In a wireless ad hoc network, each mobile node has a transmission range and energy cost. A node v can receive the signal from another node u iff node v is within node u's transmission range. We assume that when node u received a message and then forwarded the message to another node, it would consume node u some energy, which will be categorized as the cost of node u forwarding the data for other nodes. If the receiving node is not within the sender's transmission range, then it must choose some intermediate nodes to relay the message. So unlike wired networks, all nodes in the wireless ad hoc network should be able to act as a router. On the other hand, the wireless node usually uses omnidirectional antennas, which means that it can use a broadcasting-like manner to distribute the message to all nodes within its transmission range.

Usually, there are two different categories of wireless ad hoc nodes: *fixed* transmission range and *adjustable* transmission range. For *fixed* transmission range nodes, their transmission range has been fixed and cannot be adjusted afterward. So there is a directed arc from u to v if node v is within the transmission range of node u. Here the transmission cost depends on node u, regardless of the distance between two nodes. Thus, the wireless ad hoc network can be considered a *node weighted graph*, where the weight of each node is its cost to forward a unit data. If all nodes' transmission range is the same, by properly scaling, we can assume all nodes have transmission range 1. Thus, wireless topology can be modeled by a *unit disk graph (UDG)*.

The second type of wireless network is that each wireless node can adjust its transmission range: they can adjust their transmission power to the amount needed to reach the next relay node. The power needed to send a packet from node u to v consists of three parts. First, the source node u needs to consume some power to prepare the packet. Second, node u needs to consume some power to send the message to v. The power required to support the transmission between u and v not only depends on u, but also depends on the geometry distance of u and v. In the literature, it is often assumed that the power needed to support a link uv is $d_u \cdot |uv|^\beta$, where $2 \leq \beta \leq 5$ depends on the transmission environment, $|uv|$ is the Euclidean distance between u and v, and d_u is a positive number depending on node u only. Finally, when v receives the packet, it needs to consume some power to receive, store, and then process that packet. Thus, the weight of an edge uv is the power consumed for transmitting packet from u to v plus some possible energy consumed by u and v to process the signal. The wireless network under this model can be considered a *link weighted graph*: all wireless devices are the vertices of the graph, and the weight of each link uv is the total energy cost of communication using link uv.

5.2.2 Algorithm Mechanism Design

In designing efficient, centralized (with input from individual agents) or distributed algorithms and network protocols, the computational agents are typically assumed to be either *correct/obedient* or *faulty* (also called adversarial). Here, agents are said to be *correct/obedient* if they follow the protocol correctly; agents are said to be *faulty* if (1) they stop working, or (2) they drop messages, or (3) they act arbitrarily, which is also called *Byzantine failure* (i.e., they may deviate from the protocol in arbitrary ways that harm other users, even if the deviant behavior does not bring them any obvious tangible benefits).

In contrast, as mentioned before, economists design market mechanisms in which it is assumed that agents are *rational*. The rational agents respond to well-defined incentives and will deviate from the protocol only if it improves its gain. A rational agent is neither correct/obedient nor adversarial.

A standard economic model for analyzing scenarios in which the agents act according to their own self-interests is as follows. There are n agents. Each agent i, for $i \in \{1, \cdots, n\}$, has some private information t^i, called its *type*. The type t^i could be its cost to forward a packet in a network environment, or it could be its willing payment for a good in an auction environment. Then the set of n agents define a type vector $t = (t^1, t^2, \cdots, t^n)$, which is called the *profile*. There is an output specification that maps each type vector t to a set of allowed outputs. Agent i's preferences over the possible outputs are given by a valuation function v^i that assigns a real number $v^i(t^i, o)$ to each possible output o. Here, notice that the valuation of an agent does not depend on other agents' types. Everything in the scenario is public knowledge except the type t^i, which is a private information to agent i.

Definition 5.1 A Mechanism $M = (A, \mathcal{O}, p)$ defines three functions: a set of strategies A for all agents, an output function $\mathcal{O}$, and a payment function $p = (p^1, \ldots, p^n)$:

1. *For each agent i, it has a set of strategies A^i. Agent i can only choose a strategy $a \in A^i$.*
2. *For each strategy vector $a = (a^1, \cdots, a^n)$, i.e., the agent i plays a strategy $a^i \in A^i$, the mechanism computes an* output $o = \mathcal{O}(a^1, \cdots, a^n)$ *and a* payment $p^i = p^i(a)$. *Here, the payment p^i is the money given to each participating agent i. If $p^i < 0$, it means that the agent has to pay $-p^i$ to participate in the action.*

For an agent i, given the output o and the payment p^i, its utility is $u^i(o, t^i) = v^i(t^i, o) + p^i$. If a strategy a^i by an agent i is *dominant*, the agent i maximizes its utility *regardless* of whatever other agents do. Considering all different strategies, there will be too many candidate mechanisms; but with the Revelation Principle, we only need to focus our attention on these *direct revelation mechanisms*. A mechanism is a *direct revelation mechanism* if the types are the strategy space A^i. In this chapter, we only consider the direct revelation mechanisms. In practice, a mechanism should satisfy the following properties:

1. **Incentive Compatibility (IC).** The payment function should satisfy the *incentive compatibility*; that is, for each agent i,

$$v^i(t^i, o(a^{-i}, t^i)) + p^i(a^{-i}, t^i) \geq v^i(t^i, o(a^{-i}, a^i)) + p^i(a^{-i}, a^i).$$

   In other words, revealing the type t^i is the *dominating strategy*. If the payment were computed by a strategyproof mechanism, he would have no incentive to lie about its type because his overall utility would be not greater than it would have been if he had told the truth.
2. **Individual Rationality (IR).** It is also called *Voluntary Participation*. For each agent i and any a^{-i}, it should have non-negative utilities. That is, if agent i reveals its true type t^i, then its utility should be non-negative.
3. **Polynomial Time Computability (PC).** All computation is done in polynomial time. Notice that after every agent declares its type, the mechanism must compute an output o and a payment vector to all agents. For example, for the family of VCG mechanisms, the output that maximizes the summation of the valuations of all nodes must be found. When the optimal output cannot be found exactly, the individual agent may have incentives to misreport its type initially.

A mechanism is *strategyproof* or *truthful* if it satisfies both IR and IC properties. In the remainder of this chapter, we focus attention on these *truthful* mechanisms only.

Arguably the most important positive result in mechanism design is what is usually called the generalized Vickrey-Clarke-Groves (VCG) mechanism by Vickrey,[18] Clarke,[5] and Groves.[9] The VCG mechanism applies to mechanism design maximization problems where the objective function is simply the sum of all agents' valuations and the set of possible outputs is assumed to be finite.

A maximization mechanism design problem is called *utilitarian* if its objective function satisfies that $g(o,t) = \sum_i v^i(t^i, o)$. A direct revelation mechanism $m = (o(t), p(t))$ belongs to the VCG family if (1) the output $o(t)$ computed based on the type vector t maximizes the objective function $g(o,t) = \sum_i v^i(t^i, o)$, and (2) the payment to agent i is

$$p^i(t) = \sum_{j \neq i} v^j(t^j, o(t)) + h^i(t^{-i}).$$

Here, $h^i()$ is an arbitrary function of t^{-i} and a different agent could have different function $h^i()$ as long as it is defined on t^{-i}. It is proved by Groves[9] that a VCG mechanism is truthful. Green and Laffont[8] proved that, under mild assumptions, VCG mechanisms are the only truthful implementations for utilitarian problems.

An output function of a VCG mechanism is required to maximize the objective function. This makes the mechanism computationally intractable in many cases. Notice that replacing the optimal algorithm with a non-optimal approximation usually leads to untruthful mechanisms. In this chapter, we study how to perform truthful routing for multicast, which is known to be NP-hard and thus VCG mechanisms cannot be applied.

5.2.3 Priori Arts

There are generally two ways to implement the truthful computing: (1) credit-based method and (2) incentive-based method.

The first category uses various non-monetary approaches, including auditing, systemwide optimal point analysis, and some hardwares. Credit-based methods have been studied for several years, and most of them are based on simulation and are heuristic.

Nodes that agree to relay traffic but do not are termed "misbehaving." Marti et al.[13] used *Watchdog* and *Pathrater* to identify misbehaving users and avoid routing through these nodes. *Watchdog* runs on every node, keeping track of how the other nodes behave; *Pathrater* uses this information to calculate the route with the highest reliability. Notice that this method ignores the reason why a node refused to relay the transit traffics for other nodes. A node will be wrongfully labeled as misbehaving when its battery power cannot support many relay requests and thus refuses to relay. It also does not provide any incentives to encourage nodes to relay the message for other nodes.

Buttyan and Hubaux[3] focused on the problem of how to stimulate selfish nodes to forward the packets for other nodes. Their approach is based on a so-called *nuglet counter* in each node. A node's counter is decreased when sending its own packet, and is increased when forwarding other nodes' packets. All counters should always remain positive. In order to protect the proposed mechanism against misuse, they presented a scheme based on a trusted and tamper-resistant hardware module in each node that generates cryptographically protected security headers for packets and maintains the nuglet counters of the nodes. They also studied the behavior of the proposed mechanism analytically and by means of simulations, and showed that it indeed stimulates the nodes for packet forwarding.

They still use a nuglet counter to store the nuglets and they use a fine that decreases the nuglet counter to prevent the node from not relaying the packet. They use the *packet purse model* to discourage the user from sending useless traffic and overloading the network. The basic idea presented in Ref. 4 is similar to Ref. 3 but different in the implementation.

Srinivasan et al.[16] proposed two acceptance algorithms. These algorithms are used by the network nodes to decide whether to relay traffic on a per-session basis. The goal is to balance* the energy consumed by a node in relaying traffics for others with energy consumed by other nodes to relay its traffic and to find an optimal trade-off between energy consumption and session blocking probability. By taking decisions

*It is impossible to strictly balance the number of packets a node has relayed for other nodes and the number of packets of this node relayed by other nodes because, in a wireless ad hoc network, the majority of packet transmissions are relayed packets. For example, consider a path of h hops. $h - 1$ nodes on the path relay the packets for others. If the average path length of all routes is h, then $1 - 1/h$ fractions of the transmissions are transit traffic.

on a per session basis, the per packet processing overhead of previous schemes is eliminated. In Ref. 17, a distributed and scalable acceptance algorithm called GTFT is proposed. They proved that GTFT results in Nash equilibrium and the system converges to the rational and optimal operating point. Notice that they assumed that each path is h hops long and the h relay nodes are chosen with equal probability from the remaining $n - 1$ nodes, which may be unrealistic.

Salem et al.[15] presented a charging and rewarding scheme for packet forwarding in multihop cellular networks. In their network model, there is a base station to forward the packets. They use symmetric cryptography to cope with the lying. To count several possible attacks, it precharges some nodes and then refunds them only if a proper acknowledgment is received. Their basic payment scheme is still based on nuglets.

Jakobsson et al.[11] described an architecture for fostering collaboration between selfish nodes of multihop cellular networks. Based on this architecture, they provided mechanisms based on per-packet charge to encourage honest behavior and to discourage dishonest behavior. In their approach, all packet originators attach a payment token to each packet, and all intermediaries on the packet's path to the base station verify whether this token corresponds to a special token called *winning ticket*. Winning tickets are reported to nearby base stations at regular intervals. The base stations, therefore, receive both reward claims (which are forwarded to some accounting center) and packets with payment tokens. After verifying the validity of the payment tokens, base stations send the packets to their desired destinations, over the backbone network. The base stations also send the payment tokens to an accounting center. Their method also involves some traditional security methods, including auditing, node abuse detection and encryption, etc.

The *incentive-based methods* borrow some ideas from the micro-economic and game-theoretic world, which involve monetary transfer. The key result of this category is that all nodes will not deviate from their normal activities because they will benefit most when they reveal their true cost, even knowing all other nodes' true costs. We can thus achieve the optimal system performance. This idea has been introduced by Nisan and Ronen[14] and is known as the algorithm mechanism design.

Nisan and Ronen[14] provided a polynomial-time strategyproof mechanism for optimal unicast route selection in a centralized computational model. In their formulation, the network is modeled as an abstract graph $G = (V, E)$. Each edge e of the graph is an agent and has a private type t_e, which represents the cost of sending a message along this edge. The mechanism-design goal is to find a Least Cost Path (LCP) $\mathsf{LCP}(x, y)$ between two designated nodes x and y. The valuation of an agent e is $-t_e$ if the edge e is part of the path $\mathsf{LCP}(x, y)$ and 0 otherwise. Nisan and Ronen used the VCG mechanism for payment. The payment to an agent e is $D_{G-\{e\}}(x, y) - D_G(x, y)$, where $D_{G-\{e\}}(x, y)$ is the cost of the LCP through G when edge e is not presented and $D_G(x, y)$ is the cost of the least cost path $\mathsf{LCP}(x, y)$ through G. Clearly, there must be two link disjoint paths connecting x and y to prevent the monopoly. The result can be easily extended to deal with wireless unicast problems for an arbitrary pair of terminals.

Feigenbaum et al.[6] then addressed truthful low-cost routing in a different network model. They assume that each node k incurs a transit cost c_k for each transit packet it carries. For any two nodes i and j of the network, $T_{i,j}$ is the intensity of the traffic (number of packets) originating from i and destined for node j. Their strategyproof mechanism again is essentially the VCG mechanism. They gave a distributed method such that each node i can compute a payment $p_{ij}^k > 0$ to node k for carrying the transit traffic from node i to node j if node k is on the LCP $\mathsf{LCP}(i, j)$. Anderegg and Eidenbenz [1] recently proposed a similar routing protocol for wireless ad hoc networks based on the VCG mechanism again. They assumed that each link has a cost and each node is a selfish agent.

For multicast flow, Feigenbaum et al.[7] assumed that there is *fixed* multicast infrastructure, given any set of receivers $Q \subset V$, that connects the source node to the receivers. Additionally, for each user $q_i \in Q$, they assumed a *fixed* path from the source to it, determined by the multicast routing infrastructure. Then for every subset R of receivers, the delivery tree $T(R)$ is merely the union of the fixed paths from the source to the receivers R. They also assumed that there is a link cost associated with each communication link in the network and the link cost is *known* to everyone. For each receiver q_i, there is a valuation w_i that this user values the reception of the data from the source. This information w_i is only known to q_i. User q_i will report a number w_i', which is the amount of money he is willing to pay to receive the data. The source node then

selects a subset $R \subset Q$ of receivers to maximize the difference $\sum_{i \in R} w_i' - C(R)$, where $C(R)$ is the cost of the multicast tree $T(R)$ to send data to all nodes in R. The approach of fixing the multicast tree is relatively simple to implement but could not model the greedy nature of all network terminals in the network.

5.2.4 Problem Statement and Network Model

In this chapter, we consider a wireless ad hoc network composed of n selfish nodes $V = \{v_1, v_2, \cdots, v_n\}$. Every node v_i has a fixed transmission range r_i, and nodes u and v can communicate with each other if and only if $|v_i v_j| \leq \min\{r_i, r_j\}$. When node v_i sends a packet to one of its neighbors, say v_j, all v_i's neighbors can receive this packet. Thus, every node will broadcast its packet. We assume each node v_i has a private cost c_i to broadcast a unit data (the unit data could be 1 byte or 1 Megabyte). In this chapter, we model this wireless ad hoc network as a node weighted graph $G = (V, E, c)$, where V is the set of wireless nodes and $e = v_i v_j \in E$ if and only if v_i and v_j can communicate with each other directly. Here, $c = \{c_1, c_2, \cdots, c_n\}$ is the cost profile of all nodes. Notice here that the graph is undirected.

Based on the node weighted graph, we now define the multicast problem as follows. Given a set of receivers $Q = \{q_0, q_1, q_2, \cdots, q_{r-1}\} \subset V$, when selecting node $q_i \in Q$ as the source, the multicast problem is to find a tree $T \subset G$ spanning all receiving terminals Q. For simplicity, we assume that q_0 is the source of the multicast. Each node v_i is required to declare a cost d_i of relaying the message. Based on the declared cost profile $d = \{d_1, d_2, \cdots, d_n\}$, the source node constructs the multicast tree and decides the payment for each node. It is well known[10,12] that it is NP-hard to find the minimum-cost multicast tree when given an arbitrary node weighted graph G, and it is at least as hard to approximate as the set cover problem. Klein and Ravi[12] showed that it can be approximated within $O(\ln r)$, where r is the number of receivers. The utility of an agent is its payment received, minus its cost if it is selected in the multicast tree. Instead of reinventing the wheel, we will still use the previously proposed structures for multicast as the output of our mechanism. Given a multicast tree, we will study the design of strategyproof payment schemes based on these trees.

Given a graph G, we use $\omega(G)$ to denote the total cost of all nodes in this network. If we change the cost of any agent i (link e_i or node v_i) to c_i', we denote the new network as $G' = (V, E, c|^i c_i')$, or simply $c|^i c_i'$. If we remove one agent i from the network, we denote it as $c|^i \infty$. Denote $G \backslash e_i$ as the network without link e_i, and denote $G \backslash v_i$ as the network without node v_i and all its incident links. For simplicity of notation, we use the cost vector c to denote the network $G = (V, E, c)$ if no confusion is caused.

5.3 Strategyproof Multicast

In this section, we discuss in detail how to conduct truthful multicast when the network is modeled by a node weighed communication graph. We specifically study the following three structures: (1) least cost path star (LCPS), (2) virtual minimum spanning tree (VMST), (3) and node weighted Steiner tree (NST). In practice, for various applications, receivers and senders in the same multicast group usually belong to the same organization or company, so their behavior can be expected to be cooperative instead of uncooperative. Thus, we assume that every receiver will relay the packet for other receivers for free.

5.3.1 Strategyproof Mechanism Based on LCPS

5.3.1.1 Least Cost Path Star (LCPS)

Given a network modeled by the graph G, a source node s, and a set of r receivers Q, the least cost path star is the union of all r shortest paths from the source to each of the receivers in Q. In practice, this is one of the most widely used methods of constructing the multicast tree because it takes advantage of the unicast routing information collected by the distance-vector algorithm or link-state algorithm. Notice that, although here we only discuss the use of least cost path star for the node weighted case, all results presented in this subsection can be extended to the link weighted scenario without any difficulty, when each link will incur a cost when transmitting data.

5.3.1.2 Constructing LCPS

For each receiver $q_i \neq s$, we compute the shortest path (least cost path), denoted by $\mathsf{LCP}(s, q_i, d)$, from the source s to q_i under the reported cost profile d. The union of all least cost paths from the source to receivers is called *least cost path star*, denoted by *LCPS(d)*. Clearly, we can construct LCPS in time $O(n \log n + m)$. The remaining issue is how to design a truthful payment scheme while using LCPS as output.

5.3.1.3 VCG Mechanism on LCPS Is Not Strategyproof

Intuitively, we would like to use the VCG payment scheme in conjunction with the LCPS tree structure as follows. The payment $p_k(d)$ to every node v_k is

$$p_k(d) = \omega(LCPS(d\ |^k\ \infty)) - \omega(LCPS(d)) + d_k.$$

We show by example that the above payment scheme is not strategyproof. Figure 5.1 illustrates such an example where node V_2 will have a negative utility when it reveals its true cost.

Notice that $\omega(LCPS(c)) = 2M + \epsilon$ and $\omega(LCPS(c|^1\infty)) = M + \epsilon$. If v_1 reveals its true cost, its payment is $p_1(c) = \omega(LCPS(c|^1\infty)) - \omega(LCPS(c)) + M = M + \epsilon - (2M + \epsilon) + M = 0$. Thus, its utility is $p^1(c) - C_1 = 0 - M < 0$, which violates the IR property.

5.3.1.4 Strategyproof Mechanism on LCPS

Now we describe our strategyproof mechanism that does not rely on VCG payment. For each receiver $q_i \neq s$, we compute the least cost path from the source s to q_i, and compute a payment $p_k^i(d)$ to every node v_k on the $\mathsf{LCP}(s, q_i, d)$ using the scheme for unicast

$$p_k^i(d) = d_k + |\mathsf{LCP}(s, q_i, d\ |^k\ \infty)| - |\mathsf{LCP}(s, q_i, d)|.$$

Here, $|\mathsf{LCP}(s, q_i, d)|$ denotes the total cost of the least cost path $\mathsf{LCP}(s, q_i, d)$. The payment $p_k^i(d) = 0$ if node v_k is not on $\mathsf{LCP}(s, q_i, d)$. The total payment to a link v_k is then

$$p_k(d) = \max_{q_i \in Q} p_k^i(d). \tag{5.1}$$

Theorem 5.1 *Payment (5.1) based on LCPS is truthful and it is minimum among all truthful payments based on LCPS.*

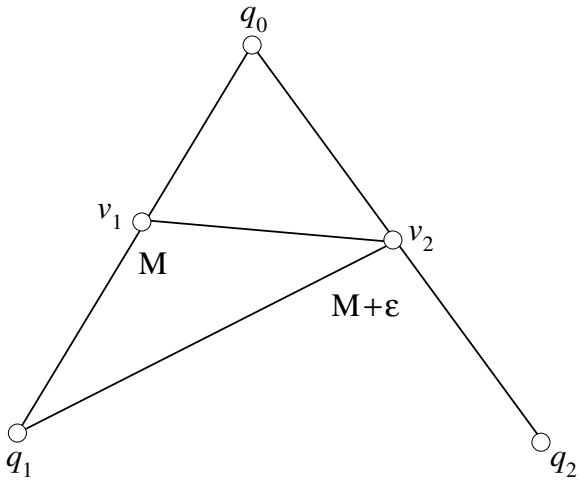


FIGURE 5.1 The cost of terminals are $v_1 = M$ and $v_2 = M + \epsilon$.

Proof Clearly, when node v_k reports its cost truthfully, it has non-negative utility; that is, the payment scheme satisfies the IR property. In addition, because the payment scheme for unicast is truthful, v_k cannot lie its cost to increase its payment $p_k^i(c)$ based on $\mathsf{LCP}(s, q_i, d)$. Thus, it cannot increase $\max_{q_i \in Q} p_k^i(d)$ by lying its cost. In other words, our payment scheme is truthful.

We then show that the above payment scheme pays the minimum among all strategyproof mechanisms using LCPS as the output. Before showing the optimality of our payment scheme, we give some definitions first. Consider all paths from source node s to a receiver q_i; they can be divided into two categories: with node v_k or without node v_k. The path with the minimum length among these paths with edge v_k is denoted as $\mathsf{LCP}_{v_k}(s, q_i, d)$; and the path with the minimum length among these paths without edge v_k is denoted $\mathsf{LCP}_{-v_k}(s, q_i, d)$.

Assume that there is another payment scheme $\tilde{p}$ that pays less for a node v_k in a network G under a cost profile d. Let $\delta = p_k(d) - \tilde{p}_k(d)$; then $\delta > 0$. Without loss of generality, assume that $p_k(d) = p_k^i(d)$. Thus, node v_k is on $\mathsf{LCP}(s, q_i, d)$ and the definition of $p_k^i(d)$ implies that

$$|\mathsf{LCP}_{-v_k}(s, q_i, d)| - |\mathsf{LCP}(s, q_i, d)| = p_k(d) - d_k.$$

Then consider another cost profile $d' = d \mid^k (p_k(d) - \frac{\delta}{2})$, where the true cost of node v_k is $p_k(d) - \frac{\delta}{2}$. Under profile d', since $|\mathsf{LCP}_{-v_k}(s, q_i, d')| = |\mathsf{LCP}_{-e_k}(s, q_i, d)|$, we have

$$\begin{aligned}
|\mathsf{LCP}_{v_k}(s, q_i, d')| &= |\mathsf{LCP}_{v_k}(s, q_i, d \mid^k 0)| + p_k(d) - \frac{\delta}{2} \\
&= |\mathsf{LCP}_{v_k}(s, q_i, d)| + p_k(d) - \frac{\delta}{2} - d_k \\
&= |\mathsf{LCP}(s, q_i, d)| + p_k(d) - \frac{\delta}{2} - d_k \\
&= |\mathsf{LCP}_{-v_k}(s, q_i, d)| - \frac{\delta}{2} \\
&< |\mathsf{LCP}_{-v_k}(s, q_i, d)| = |\mathsf{LCP}_{-v_k}(s, q_i, d')|.
\end{aligned}$$

Thus, $v_k \in LCPS(d')$. From the following Lemma 5.1, we know that the payment to node v_k is the same for cost profile d and d'. Thus, the utility of node v_k under the profile d' by the payment scheme $\tilde{p}$ becomes $\tilde{p}_k(d') - c_k = \tilde{p}_k(d) - c_k = \tilde{p}_k(d) - (p_k(d) - \frac{\delta}{2}) = -\frac{\delta}{2} < 0$. In other words, under the profile d', when the node e_k reports its true cost, it gets a negative utility under payment scheme $\tilde{p}$. Thus, $\tilde{p}$ is not strategyproof. This finishes our proof. □

Lemma 5.1 *If a mechanism with output T and the payment function $\tilde{p}$ is truthful, then for every node v_k in network, if $v_k \in T$, then payment function $\tilde{p}_k(d)$ should be independent of d_k.*

Proof We prove it by contradiction. Suppose that there exists a truthful payment scheme such that $\tilde{p}_k(d)$ depends on d_k. There must exist two valid declared costs x_1 and x_2 for node v_k such that $x_1 \neq x_2$ and $\tilde{p}_k(d \mid^k x_1) \neq \tilde{p}_k(d \mid^k x_2)$. Without loss of generality we assume that $\tilde{p}_k(d \mid^k x_1) > \tilde{p}_k(d \mid^k x_2)$. Now consider the situation when node v_k has an actual cost $c_k = x_2$. Obviously, node v_k can lie its cost as x_2 to increase his utility, which violates the incentive compatibility (IC) property. This finishes the proof. □

Notice that the payment based on $p_k(d) = \min_{q_i \in Q} p_k^i(d)$ is not truthful because a node can lie its cost upward so it can discard some low payment from some receiver. In addition, the payment $p_k(d) = \sum_{q_i \in Q} p_k^i(d)$ is *not* truthful either.

5.3.2 Strategyproof Mechanism Based on VMST

5.3.2.1 Constructing VMST

We first describe our method to construct the virtual minimum spanning tree.

Algorithm 5.1

Virtual MST Algorithm.

1. First calculate the pairwise least cost path $\mathsf{LCP}(q_i, q_j, d)$ between any two terminals $q_i, q_j \in Q$ when the cost vector is d.
2. Construct a virtual complete link weighted network $K(d)$ using Q (including the source node here) as its terminals, where the link q_iq_j corresponds to the least cost path $\mathsf{LCP}(q_i, q_j, d)$, and its weight $w(q_iq_j)$ is the cost of the path $\mathsf{LCP}(q_i, q_j, d)$, i.e., $w(q_iq_j) = |\mathsf{LCP}(q_i, q_j, d)|$.
3. Build the minimum spanning tree (MST) on $K(d)$. The resulting MST is denoted as $VMST(d)$.
4. For every virtual link q_iq_j in $VMST(d)$, we find the corresponding least cost path $\mathsf{LCP}(q_i, q_j, d)$ in the original network. Combining all these paths can generate a subgraph of G, say $VMSTO(d)$.
5. All nodes on $VMSTO(d)$ will relay the packets.

Notice that a terminal v_k is on $VMSTO(d)$ iff v_k is on some virtual links in the $VMST(d)$, so we can focus our attention on these terminals in $VMST(d)$. It is not very difficult to show that the cost of VMST could be very large compared to the optimal. But when all nodes have the same transmission ranges in the original wireless ad hoc network, which can be modeled as UDG, the following theorem shows that the virtual minimum spanning tree can approximate the cost of the optimal tree within a constant factor.

Theorem 5.2 *$VMST(G)$ is a 5-approximation of the optimal solution in terms of the total cost if the wireless ad hoc network is modeled by a unit disk graph.*

Proof Assume that the optimal solution is a tree called T_{opt}. Let $V(T_{opt})$ be the set of nodes used in the tree T_{opt}. Clearly, $\omega(T_{opt}) = \sum_{v_i \in V(T_{opt})} c_i$. Similarly, for any spanning tree T of $K(G, Q)$, we define $\omega(T) = \sum_{e \in T} w(e)$. Following we will prove $5 \cdot \omega(T_{opt}) \geq \omega(VMST(G))$.

First, for all nodes in T_{opt}, when disregarding the node weight, there is a spanning tree T'_{opt} on $V(T_{opt})$ with node degree at most 5 because the wireless network is modeled by a unit disk graph. This is due to the well-known fact that there is a Euclidean minimum spanning tree with the maximum node degree at most 5 for any set of two-dimensional points. Note here that we only need to know the existence of T'_{opt}; we do not need to construct such a spanning tree explicitly. Obviously, $\omega(T_{opt}) = \omega(T'_{opt})$. Thus, tree T'_{opt} is also an optimal solution with maximal node degree of at most 5.

For spanning tree T'_{opt}, we root it at an arbitrary node and duplicate every link in T'_{opt} (the resulting structure is called DT'_{opt}). Clearly, every node in DT'_{opt} has even degree now. Thus, we can find a Euler circuit, denoted by $EC(DT'_{opt})$, that visits every vertex of DT'_{opt} and uses every edge of DT'_{opt} exactly once, which is equivalent to saying that every edge in $T'_{opt}(G)$ is used exactly twice. Consequently, we know that every node v_k in $V(T_{opt})$ is used exactly $deg_{T'_{opt}}(v_k)$ times. Here, $deg_G(v)$ denotes the degree of a node v in a graph G. Thus, the total weight of the Euler circuit is at most 5 times the weight $\omega(T'_{opt})$; that is,

$$\omega(EC(DT'_{opt})) \leq 5 \cdot \omega(T'_{opt}).$$

Notice that here if a node v_k appears multiple times in $EC(DT'_{opt})$, its weight is also counted multiple times in $\omega(EC(DT'_{opt}))$.

If we walk along $EC(DT'_{opt})$, we visit all receivers, and the length of any subpath between receivers q_i and q_j is no smaller than $|\mathsf{LCP}(q_i, q_j, G)|$. Thus, the cost of $EC(DT'_{opt})$ is at least $\omega(VMST(G))$ because $VMST(G)$ is the minimum spanning tree spanning all receivers and the cost of the edge q_iq_j in $VMST(G)$ corresponds to the path with the least cost $|\mathsf{LCP}(q_i, q_j, G)|$. In other words,

$$\omega(EC(DT'_{opt})) \geq \omega(VMST(G)).$$

Consequently, we have

$$\omega(VMST(G)) \leq \omega(EC(DT'_{opt})) \leq 5 \cdot \omega(T'_{opt}).$$

This finishes the proof. □

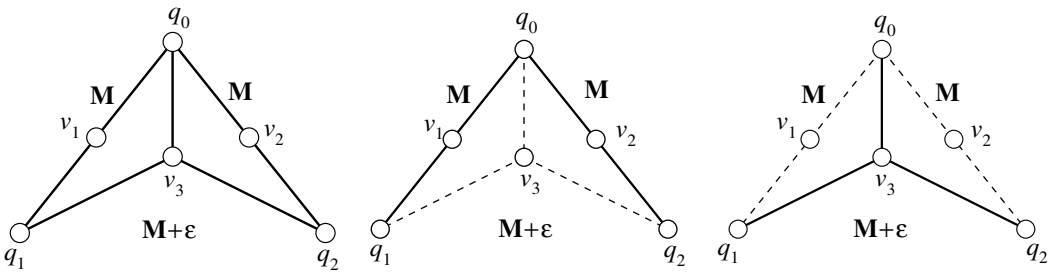


FIGURE 5.2 The cost of terminals are $c_4 = c_5 = M$ and $c_3 = M + \epsilon$.

5.3.2.2 VCG Mechanism on VMST Is Not Strategyproof

In this subsection, we show that a simple application of VCG mechanism on VMST is not strategyproof. Figure 5.2 illustrates such an example where terminal v_3 can lie its cost to improve its utility when the output is VMST. The payment to terminal v_3 is 0 and its utility is also 0 if it reports its cost truthfully. The total payment to terminal v_3 when v_3 reported a cost $d_3 = M-\epsilon$ is $\omega(VMST(c|^3\infty)) - \omega(VMST(c|^3 d_3)) + d_3 = 2M-(M-\epsilon)+M-\epsilon = 2M$ and the utility of terminal v_3 becomes $u_3(c|^3 d_3) = 2M-(M+\epsilon) = M-\epsilon$, which is larger than $u_3(c) = 0$. Thus, the VCG mechanism based on VMST is not strategyproof.

5.3.2.3 Strategyproof Mechanism on VMST

Before discussing the strategyproof mechanism based on VMST, we give some related definitions first. Given a spanning tree T and a pair of terminals p and q on T, clearly there is a unique path connecting them on T. We denote such path as $\Pi_T(p,q)$, and the edge with the maximum length on this path as $LE(p,q,T)$. For simplicity, we use $LE(p,q,d)$ to denote $LE(p,q,VMST(d))$ and use $LE(p,q,d\ |^k\ d'_k)$ to denote $LE(p,q,VMST(d\ |^k\ d'_k))$.

Following is our truthful payment scheme when the output is the multicast tree $VMST(d)$.

Algorithm 5.2

Truthful payment scheme based on VMST.

1. For every terminal $v_k \in V \backslash Q$ in G, first calculate $VMST(d)$ and $VMST(d\ |^k\ \infty)$ according to the terminals' declared costs vector d.
2. For any edge $e = q_i q_j \in VMST(d)$ and any terminal $v_k \in \mathsf{LCP}(q_i, q_j, d)$, we define the payment to terminal v_k based on the virtual link $q_i q_j$ as follows:

$$p_k^{ij}(d) = |LE(q_i, q_j, d\ |^k\ \infty)| - |\mathsf{LCP}(q_i, q_j, d)| + d_k.$$

Otherwise, $p_{ij}^k(d)$ is 0. The final payment to terminal v_k based on $VMST(d)$ is

$$p_k(d) = \max_{q_i q_j \in VMST(d)} p_k^{ij}(d). \tag{5.2}$$

Theorem 5.3 *Our payment scheme (5.2) is strategyproof and minimum among all truthful payment schemes based on VMST structure.*

Instead of proving Theorem 5.3, we prove Theorem 5.4, Theorem 5.5, and Theorem 5.6 in the remainder of this subsection.

Before the proof of Theorem 5.3, we give some related notations and observations. Considering the graph $K(d)$ and a node partition $\{Q_i, Q_j\}$ of Q, if an edge's two end-nodes belong to different node sets of the partition, we call it a *bridge*. All bridges $q_s q_t$ over node partition Q_i, Q_j in the graph $K(d)$ satisfying $v_k \notin \mathsf{LCP}(q_s, q_t, d)$ form a bridge set $B^{-v_k}(Q_i, Q_j, d)$. Among them, the bridge with the minimum length is denoted as $MB^{-v_k}(Q_i, Q_j, d)$ when the nodes' declared cost vector is d. Similarly, all bridges $q_s q_t$ over node partition Q_i, Q_j in the graph $K(d)$ satisfying $v_k \in \mathsf{LCP}(q_s, q_t, d)$ form a bridge set $B^{v_k}(Q_i, Q_j, d)$. The bridge in $B^{v_k}(Q_i, Q_j, d)$ with the minimum length is denoted as $BM^{v_k}(Q_i, Q_j, d)$. Obviously, we have

$$BM(Q_i, Q_j, d) = \min\{BM^{v_k}(Q_i, Q_j, d), BM^{-v_k}(Q_i, Q_j, d)\}.$$

We then state our main theorems for the payment scheme discussed above.

Theorem 5.4 *Our payment scheme satisfies IR.*

Proof First of all, if terminal v_k is not chosen as a relay terminal, then its payment $p_k(d \mid^k c_k)$ is clearly 0 and its valuation is also 0. Thus, its utility $u_k(d \mid^k c_k)$ is 0.

When terminal v_k is chosen as a relay terminal when it reveals its true cost c_k, we have $|LE(q_i, q_j, d \mid^k \infty)| \geq |\mathsf{LCP}(q_i, q_j, d \mid^k c_k)|$. This is due to the following observation: for any cycle C in a graph G, assume e_c is the longest edge in the cycle; then $e_c \notin MST(G)$. The lemma immediately follows from

$$p_{ij}^k(d \mid^k c_k) = |LE(q_i, q_j, d \mid^k \infty)| - |\mathsf{LCP}(q_i, q_j, d \mid^k c_k)| + c_k > c_k.$$

This finishes the proof. □

From the definition of the incentive compatibility (IC), we assume that the d_{-k} is fixed throughout the proof. For our convenience, we will use $G(d_k)$ to represent the graph $G(d \mid^k d_k)$. We first prove a series of lemmas that will be used to prove that our payment scheme satisfies IC.

Lemma 5.2 *If $v_k \in q_i q_j \in VMST(d)$, then $p_k^{ij}(d)$ does not depend on d_k.*

Proof Remember that the payment based on a link $q_i q_j$ is $p_k^{ij}(d) = |LE(q_i, q_j, d \mid^k \infty)| - |\mathsf{LCP}(q_i, q_j, d)| + d_k$. The first part $LE(q_i, q_j, d \mid^k \infty)$ is the longest edge of the unique path from q_i to q_j on tree $VMST(d \mid^k \infty)$. Clearly, it is independent of d_k. Now consider the second part $\mathsf{LCP}(q_i, q_j d) - d_k$. From the assumption, we know that $v_k \in \mathsf{LCP}(q_i, q_j, d)$, so the path $\mathsf{LCP}(q_i, q_j, d)$ remains the same regardless of v_k's declared cost d_k. Thus, the summation of all terminals' cost on $\mathsf{LCP}(q_i, q_j, d)$ except terminal v_k equals

$$|\mathsf{LCP}(q_i, q_j, d \mid^k 0)| = |\mathsf{LCP}(q_i, q_j, d)| - d_k.$$

In other words, the second part is also independent of d_k. Now we can write the payment to a terminal v_k based on an edge $q_i q_j$ as follows:

$$p_k^{ij}(d) = |LE(q_i, q_j, d \mid^k \infty)| - |\mathsf{LCP}(q_i, q_j, d \mid^k 0)|,$$

Here, terminal $v_k \in \mathsf{LCP}(q_i, q_j, d)$ and $q_i q_j \in VMST(d)$. □

If a terminal v_k lies its cost c_k upward, we denote the lied cost as $\overline{c_k}$. Similarly, if terminal v_k lies its cost c_k downward, we denote the lied cost as $\underline{c_k}$. Let $E_k(d_k)$ be the set of edges $q_i q_j$ such that $v_k \in \mathsf{LCP}(q_i, q_j, d)$ and $q_i q_j \in VMST(d)$ when terminal v_k declares a cost d_k. From lemma 5.2, the non-zero payment to v_k is defined based on $E_k(d_k)$. The following lemma reveals the relationship between d_k and $E_k(d_k)$. The proof of the lemma is omitted due to its simplicity.

Lemma 5.3 $E_k(d_k) \leq E_k(d'_k)$.

We now state the proof that the payment scheme (5.2) satisfies IC.

Theorem 5.5 *Our payment scheme satisfies the incentive compatibility (IC).*

Proof For terminal v_k, if it lies its cost from c_k to $\overline{c_k}$, then $E_k(\overline{c_k}) \subseteq E_k(c_k)$, which implies that payment

$$\begin{aligned} p_k(d \mid^k \overline{c_k}) &= \max_{q_i q_j \in E_k(\overline{c_k})} p_k^{ij}(d \mid^k \overline{c_k}) \\ &\leq \max_{q_i q_j \in E_k(c_k)} p_k^{ij}(d \mid^k c_k) = p^k(d \mid^k c_k). \end{aligned}$$

Thus, terminal v_k will not lie its cost upward, so we focus our attention on the case when terminal v_k lies its cost downward. □

From lemma 5.3, we know that $E_k(c_k) \subseteq E_k(\underline{c_k})$. Thus, we only need to consider the payment based on edges in $E_k(\underline{c_k}) - E_k(c_k)$. For edge $e = q_i q_j \in E_k(\underline{c_k}) - E_k(c_k)$, let $q_I^k q_J^k = LE(q_i, q_j, d \mid^k \infty)$ in the spanning tree $VMST(d \mid^k \infty)$. If we remove the edge $q_I^k q_J^k$, we have a vertex partition $\{Q_I^k, Q_J^k\}$, where $q_i \in Q_I^k$ and $q_j \in Q_J^k$. In the graph $K(d)$, we consider the bridge $BM(Q_I^k, Q_J^k, d)$ whose weight is minimum when the terminal cost vector is d. There are two cases to consider about $BM(Q_I^k, Q_J^k, d)$: (1) $v_k \notin BM(Q_I^k, Q_J^k, d \mid^k c_k)$ or (2) $v_k \in BM(Q_I^k, Q_J^k, d \mid^k c_k)$. We discuss them individually.

***Case* 5.1** $v_k \notin BM(Q_I^k, Q_J^k, d \mid^k c_k)$. In this case, edge $q_I^k q_J^k$ is the minimum bridge over Q_I^k and Q_J^k. In other words, we have $|LE(q_i, q_j, |^k \infty)| \leq |\mathsf{LCP}(q_i, q_j, d \mid^k c_k)|$. Consequently

$$\begin{aligned} p_k^{ij}(d \mid^k \underline{c_k}) &= |LE(q_i, q_j, d \mid^k \infty)| - |\mathsf{LCP}(q_i, q_j, d \mid^k \underline{c_k})| + \underline{c_k} \\ &= |LE(q_i, q_j, d \mid^k \infty)| - |\mathsf{LCP}(q_i, q_j, d \mid^k c_k)| + c_k \\ &\leq c_k, \end{aligned}$$

which implies that v_k will not get benefit from lying its cost downward.

***Case* 5.2** $v_k \in BM(Q_I^k, Q_J^k, d \mid^k c_k)$. From the assumption that $q_i q_j \notin VMST(G(d \mid^k c_k))$, we know edge $q_i q_j$ cannot be $BM(Q_I^k, Q_J^k, d \mid^k c_k)$. Thus, there exists an edge $q_s q_t \neq q_i q_j$ such that $v_k \in \mathsf{LCP}(q_s, q_t, d \mid^k c_k)$ and $q_s q_t = BM(Q_I^k, Q_J^k, d \mid^k c_k)$. This guarantees that $q_s q_t \in VMST(d \mid^k c_k)$.

Obviously, $q_s q_t$ cannot appear in the same set of Q_I^k or Q_J^k. Thus, $q_I^k q_J^k$ is on the path from q_s to q_t in graph $VMST(d \mid^k \infty)$, which implies that $|\mathsf{LCP}(q_I^k, q_J^k, d \mid^k \infty)| = |LE(q_i, q_j, d \mid^k \infty)| \leq |LE(q_s, q_t, d \mid^k \infty)|$. Using lemma 5.3, we have $\mathsf{LCP}(q_s, q_t, d \mid^k \underline{c_k}) \in VMST(d \mid^k \underline{c_k}))$. Thus,

$$\begin{aligned} p_k^{ij}(d \mid^k \underline{c_k}) &= |LE(q_i, q_j, d \mid^k \infty)| - |\mathsf{LCP}(q_i, q_j, d \mid^k \underline{c_k})| + \underline{c_k} \\ &= |LE(q_i, q_j, d \mid^k \infty)| - |\mathsf{LCP}(q_i, q_j, d \mid^k c_k)| + c_k \\ &\leq |LE(q_s, q_t, d \mid^k \infty)| - |\mathsf{LCP}(q_i, q_j, d \mid^k c_k)| + c_k \\ &\leq |LE(q_s, q_t, d \mid^k \infty)| - |\mathsf{LCP}(q_s, q_t, d \mid^k c_k)| + c_k \\ &= p_k^{st}(d \mid^k c_k). \end{aligned}$$

This inequality concludes that even if v_k lies its cost downward to introduce some new edges in $E_k(\underline{c_k})$, the payment based on these newly introduced edges is not larger than the payment on some edges already contained in $E_k(c_k)$. In summary, node v_i does not have the incentive to lie its cost upward or downward, which proves the IC property.

Before proving Theorem 5.6, we prove the following lemma regarding all truthful payment schemes based on VMST.

Lemma 5.4 *If $v_k \in VMST(d \mid^k c_k)$, then as long as $d_k < p_k(d \mid^k c_k)$ and d^{-k} fixed, $v_k \in VMST(d)$.*

Proof Again, we prove it by contradiction. Assume that $v_k \notin VMST(d)$. Obviously, $VMST(d) = VMST(d \mid^k \infty)$. Assume that $p_k(d \mid^k c_k) = p_k^{ij}(d \mid^k c_k)$; that is, its payment is computed based on edge q_iq_j in $VMST(d \mid^k c_k)$. Let q_Iq_J be the $LE(q_i, q_j, d \mid^k \infty)$ and $\{Q_i, Q_j\}$ be the vertex partition introduced by removing edge q_Iq_J from the tree $VMST(d \mid^k \infty)$, where $q_i \in Q_i$ and $q_j \in Q_j$. The payment to terminal v_k in $VMST(d \mid^k c_k)$ is $p_k(d \mid^k c_k) = |\mathsf{LCP}(q_I, q_J, d \mid^k \infty)| - c_{ij}^{v_k}$, where $c_{ij}^{v_k} = |\mathsf{LCP}(q_i, q_j, d \mid^k 0|$. When v_k declares its cost as d_k, the length of the path $\mathsf{LCP}(q_i, q_j, d)$ becomes $c_{ij}^{v_k} + d_k = |\mathsf{LCP}(q_I, q_J, d \mid^k \infty)| - p_k(d \mid^k c_k) + d_k < |\mathsf{LCP}(q_I, q_J, d \mid^k \infty)|$.

Now consider the spanning tree $VMST(d)$. We have assumed that $v_k \notin VMST(d)$, that is, $VMST(d) = VMST(d \mid^k \infty)$. Thus, among the bridge edges over Q_i, Q_j, edge q_Iq_J has the least cost when the graph is $G \backslash v_k$ or $G(d \mid^k d_k)$. However, this is a contradiction to what we just proved: $|\mathsf{LCP}(q_i, q_j, d \mid^k d_k)| < |\mathsf{LCP}(q_I, q_J, d \mid^k \infty)|$. This finishes the proof. □

We are now ready to show that our payment scheme is optimal among all truthful mechanisms using VMST.

Theorem 5.6 *Our payment scheme is the minimum among all truthful payment schemes based on the VMST structure.*

Proof We prove it by contradiction. Assume that there is another truthful payment scheme, say $\mathcal{A}$, based on VMST, whose payment is smaller than our payment for a terminal v_k under a cost profile d. Assume that the payment calculated by $\mathcal{A}$ for terminal v_k is $\tilde{p}_k(d) = p_k(d) - \delta$, where $p_k(d)$ is the payment calculated by our algorithm and $\delta > 0$.

Now consider another profile $d \mid^k d'_k$, where the terminal v_k has the true cost $c_k = d'_k = p^k(d) - \frac{\delta}{2}$. From lemma 5.4, we know that v_k is still in $VMST(d \mid^k d'_k)$. Using lemma 5.1, we know that the payment for terminal v_k using algorithm $\mathcal{A}$ is $p_k(c) - \delta$, which is independent of terminal v_k's declared cost. Notice that $d_k = p_k(d) - \frac{\delta}{2} > p_k(d) - \delta$. Thus, terminal v_k has negative utility under the payment scheme $\mathcal{A}$ when node v_k reveals it true cost under cost profile $d \mid^k d'_k$, which violates the incentive compatibility (IC). This finishes the proof. □

By summarizing Theorem 5.4, Theorem 5.5, and Theorem 5.6, we obtain Theorem 5.3.

5.3.3 Strategyproof Mechanism Based on Spider

For a general node weighted network, in the worst case, the cost of the structure LCPS and VMST could be $\theta(n)$ times the cost of the optimal tree. It is known[10,12] that it is NP-hard to find the minimum cost multicast tree when given an arbitrary node weighted graph G, and it is at least as hard to approximate as the set cover problem. Klein and Ravi[12] showed that it can be approximated within $O(\ln r)$, where r is the number of receivers, which is within a small constant factor of the best achievable approximation ratio among all polynomial time computable trees if $N \neq NP$.

5.3.3.1 Constructing the Spider

Here we review the method used by Klein and Ravi[12] to find a node weighted Steiner tree (NST). Klein and Ravi used a special structure called a *spider* to approximate the optimal solution. A spider is defined as a tree having at most one node of degree more than two. Such a node (if it exists) is called the center of the spider. Each path from the center to a leaf is called a *leg*. The *cost* of a spider S is defined as the sum of the cost of all nodes in spider S, denoted as $\omega(S)$. The number of terminals or *legs* of the spider is denoted

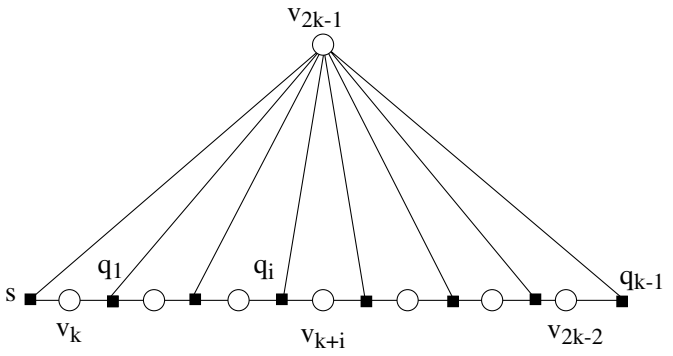


FIGURE 5.3 Terminals V_i, $1 \leq i \leq k$ are receivers; the cost of terminal v_{2k-1} is 1. The cost of each terminal $v_i, k \leq i \leq 2k_2$ is $\frac{2}{2k-i} - \epsilon$, where ϵ is a sufficiently small positive number.

by $t(S)$, and the ratio of a spider S is defined as

$$\rho(S) = \frac{\omega(S)}{t(S)}.$$

Contraction of a spider S is the operation of contracting all vertices of S to form one virtual terminal and connecting this virtual terminal to each vertex v when uv is a link before the contraction and $u \in S$. The new virtual terminal has weight zero.

Algorithm 5.3

Construct NST.

Repeat the following steps until no receivers are left and there is only one virtual terminal remaining.

1. Find the spider S with the minimum $\rho(S)$ that connects some receivers and virtual terminals.*
2. Contract the spider S by treating all nodes in it as one virtual terminal. We call this one *round*.

All nodes belong to the final unique virtual terminal to form the NST.

The following theorem is proved in Klein and Ravi.[12]

Theorem 5.7[12] *Given k receivers, the tree constructed above has cost at most* $2 \ln k$ *times of the optimal.*

5.3.3.2 VCG Mechanism on NST Is Not Strategyproof

Again, we may want to pay terminals based on the VCG scheme; that is, the payment to a terminal $v_k \in NST(d)$ is

$$p_k(d) = \omega(NST(d \mid^k \infty)) - \omega(NST(d)) + d_k.$$

We show by example that the payment scheme does *not* satisfy the IR property: it is possible that some terminal has negative utility under this payment scheme. Figure 5.3 illustrates such an example. It is not difficult to show that, in the first round, terminal v_k is selected to connect terminals s and q_1 with cost ratio $\frac{1}{k} - \frac{\epsilon}{2}$ (while all other spiders have cost ratios of at least $\frac{1}{k}$). Then, terminals s, v_k, and q_1 form a virtual

*For simplicity of the proof, we assume that there does not have to be two spiders with the same ratio. Dropping the assumption will not change our results.

terminal. At the beginning of round r, we have a virtual terminal, denoted by V_r formed by terminals v_{k+i-1}, $1 \le i \le r-1$, and receivers q_i, $1 \le i \le r$; all other receivers q_i, $r < i \le k$ are the remaining terminals. It is easy to show that we can select terminal q_{k+r-1} at round r to connect V_r and q_{r+1} with cost $\frac{1}{k+1-r} - \frac{\epsilon}{2}$. Thus, the total cost of the tree $NST(G)$ is $\sum_{i=1}^{k-1}(\frac{2}{k+1-i} - \epsilon) = 2H(k) - 2 - (k-1)\epsilon$.

When terminal v_k is not used, it is easy to see that the final tree $NST(G \backslash u_1)$ will only use the terminal v_{2k-1} to connect all receivers with cost $\frac{1}{k}$ when $\frac{1}{k-1} - \frac{\epsilon}{2} > \frac{1}{k}$. Notice that this condition can be trivially satisfied by letting $\epsilon = \frac{1}{k^2}$. Thus, the utility of terminal v_k is $p_1(d) - c(v_k) = \omega(NST(G \backslash v_k)) - \omega(NST(G)) = -2H(k) + 3 + (k-1)\epsilon$, which is negative when $k \ge 8$ and $\epsilon = 1/k^2$.

5.3.3.3 Strategyproof Mechanism on Spiders

Notice that the construction of NST is by rounds. Following, we show that if terminal v_k is selected as part of the spider with the minimum ratio under a cost profile d in a round i, then v_k is selected before or in round i under a cost profile $d' = d \mid^k d'_k$ for $d'_k < d_k$. We prove this by contradiction, which assumes that the terminal v_k will not appear before round $i+1$. Notice that the graph remains the same for round i after the profile changes, so spider $S_i(d)$ under the cost profile d is still a valid spider under the cost profile d'. Its ratio becomes $\omega_i^k(d) - d_k + d'_k < \omega_i^k(d)$, while all other spider ratios remain the same if they do not contain v_k. Thus, the spider $S_i^k(d)$ has the minimum ratio among all spiders under cost profile d', which is a contradiction. So, for terminal v_k, there exists a real value $B_k^i(d_{-k})$ such that the terminal v_k is selected before or in round i iff $d_k < B_k^i(d_{-k})$. If there are r rounds, we have an increasing sequence

$$B_k^1(d_{-k}) \le B_k^2(d_{-k}) \le \cdots \le B_k^r(d_{-k}) = B_k(d_{-k}).$$

Obviously, the terminal v_k is selected in the final multicast tree iff $d_k < B_k(d_{-k})$. Following is our payment scheme based on NST.

Definition 5.2 For a node v_k, if v_k is selected in NST, then it gets payment

$$p_k(d) = B_k(d_{-k}). \tag{5.3}$$

Otherwise, it gets payment 0.

Regarding this payment, we have the following theorem.

Theorem 5.8 *Our payment scheme (5.3) is truthful, and is minimum among all truthful payment schemes for multicast trees based on spider.*

Proof To prove that it is truthful, we prove that it satisfies IR and IC, respectively. Notice that v_k is selected iff $d_k < B_k^i(d_{-k})$, and we have $u_k(d) = B_k(d_{-k}) - d_k > 0$, which implies IR. Now we prove that our payment scheme (5.3) satisfies IC by cases. Notice that when v_k is selected, its payment does not depend on d_k, so we only need to discuss the following two cases:

***Case* 5.1** When v_k declares c_k, it is not selected. What happens if it lies its cost upward as d_k to make it not selected? From the IR property, v_k gets positive utility when it reveals its true cost, while it gets utility 0 when it lies its cost as d_k. So, it better for v_k not to lie.

***Case* 5.2** When v_k declares a cost c_k, it is not selected. What happens if it lies its cost downward as d_k to make it selected? When v_k reveals c_k, it has utility 0; after lying, it has utility $B_k(d_{-k}) - c_k$. From the assumption that v_k is not selected under cost profile $d \mid^k c_k$, we have $B_k(d_{-k}) \le c_k$. Thus, v_k will get non-positive utility if it lies, which ensures v_k revealing its true cost c_k.

So overall, v_k will always choose to reveal its actual cost to maximize its utility (IC property).

Following we prove that our payment is minimal. We prove it by contradiction. Suppose that there exists such a payment scheme $\tilde{P}$ such that for a terminal v_k under a cost profile d, the payment to $\tilde{P}_i(d)$ is smaller than our payment. Notice that in order to satisfy the IR, the terminal must be selected, so we assume $\tilde{P}_i(d) = B_k(d_{-k}) - \delta$, where δ is a positive real number. Now consider the profile $d' = d\ |^k\ (B_k(d_{-k}) - \frac{\delta}{2})$ with v_k's actual cost being $c_k = B_k(d_{-k}) - \frac{\delta}{2}$. Obviously, v_k is selected; from lemma 5.1, the payment to v_k is $B_k(d_{-k}) - \delta$. Thus, the utility of v_k becomes $u_k(d') = B_k(d_{-k}) - B_k(d_{-k}) - \delta + \frac{\delta}{2} = -\frac{\delta}{2} < 0$, which violates the IR property. This finishes our proof. □

We then study how to compute such payment to a selected node v_k. With Theorem 5.8, we only need to focus attention on how to find the value $B_k^i(d_{-k})$. Before we present our algorithm to find $B_k^i(d_{-k})$, we first review in detail how to find the minimum ratio spider. To find the spider with the minimum ratio, we find the spider centered at every vertex v_j with the minimum ratio over all vertices $v_j \in V$ and choose the minimum among them. The algorithm is as follows.

Algorithm 5.4

Find the minimum ratio spider.

Do the following process for all $v_j \in V$:

1. Calculate the shortest path tree rooted at v_j that spans all terminals. We call each shortest path a *branch*. The weight of the branch is defined as the length of the shortest path. Here, the weight of the shortest path does not include the weight of the center node v_j of the spider.
2. Sort the branches according to their weights.
3. For every pair of branches, if they have terminals in common, then remove the branch with a larger weight. Assume that the remaining branches are
$$L(v_j) = \{L_1(v_j), L_2(v_j), \cdots, L_r(v_j),\}$$
sorted in ascending order of their weights.
4. Find the minimum ratio spider with center v_j by linear scanning: the spider is formed by the first t branches such that $\frac{c_j+\Sigma_{k=1}^{i} L_k}{t} \leq \frac{c_j+\Sigma_{k=1}^{h} L_k}{h}$ for any $h \neq t$.
Assume the spider with the minimum ratio centered at terminal v_j is $S(v_j)$ and its ratio is $\rho(v_j)$.
5. The spider with minimum ratio for this graph is then $S = \min_{v_j \in V} S(v_j)$.

In Algorithm 5.5, $\omega(L_i(v_j))$ is defined as the sum of the terminals' cost on this branch, and $\Omega_i(L(v_j)) = \Sigma_{s=1}^{i}\omega(L_s(v_j)) + c_j$. If we remove node v_k, the minimum ratio spider with center v_j is denoted as $S^{-v_k}(v_j)$ and its ratio is denoted as $\rho^{-v_k}(v_j)$. Assume that $L_1^{-v_k}(v_j), L_2^{-v_k}(v_j), \cdots, L_p^{-v_k}(v_j)$ are those branches in ascending order before linear scan.

From now on, we fix d_{-k} and the graph G to study the relationship between the minimum ratio $\rho(v_j)$ of the spider centered at v_j and the cost d_k of a node v_k.

Observation 5.1 *The number of the legs of the minimum ratio spider decreases over d_k.*

If the minimum ratio spider with the terminal v_k has t legs, then its ratio will be a line with slope of $\frac{1}{t}$. So, the ratio-cost function is formed by several line segments. From the Observation 5.1, these line segments have decreasing slopes and thus they have at most r segments, where r is the number of receivers. So, given a real value y, we can find the corresponding cost of v_k in time $O(\log r)$ such that the minimum cost ratio spider $S(v_j)$ centered at node v_j has a ratio y. We the present our algorithm to find these line segments as follows.

Algorithm 5.5

Find the ratio-cost function $y = \mathcal{R}_{v_j}(x)$ over the cost x of v_k.

There are two cases here: $j = k$ or $j \neq k$.

***Case* 5.1** $j = k$, we apply the following procedures:

Apply steps 1, 2, and 3 of Algorithm 5.4 to get $L(v_k)$.
Set the number of legs to $t = 1$, lower bound $lb = 0$, and upper bound $ub = 0$.
While $t < r$ do the following:
$\{ub = (t+1) * \omega(L_{t+1}(v_k)) - \Omega_{t+1}(L(v_k))$
$y = \frac{\Omega_t(L(v_k))+x}{t}$ for $x \in [lb, ub)$
Set $lb = ub$ and $t = t + 1$ $\}$
Let $y = \frac{\Omega_r(L(v_k))+x}{r}$ for $x \in [lb, \infty)$.

***Case* 5.2** $j \neq k$, we apply the following procedures:

1. Remove terminal v_k; apply Algorithm 5.4 to find $S^{-v_k}(v_j)$.
2. Find the shortest path with terminal v_k from v_j to every receiver; sort these paths according to their length in descending order, say sequence
$$L^{v_k}(v_j) = \left\{L_1^{v_k}(v_j), L_2^{v_k}(v_j), \cdots, L_r^{v_k}(v_j)\right\}.$$
Here, r is the number of terminals and $\omega(L_i^{v_k}(v_j))$ is the sum of terminals on path $L_i^{v_k}(v_j)$ *excluding* terminal v_k.
3. Hereafter, t is the index for branches in $L^{v_k}(v_j)$ and l is the index for paths in $L^{-v_k}(v_j)$.
4. For $L_t^{v_k}(v_j)$ $(1 \leq t \leq r)$, there may exist one or more branches in $L^{-v_k}(v_j)$ such that they have common terminals with $L_t^{v_k}(v_j)$. If there is more than one such branch, choose the branch with the minimum cost, say $L_l^{-v_k}(v_j)$. We define upper bound $upper_t$ for $L_t^{v_k}(v_j)$ equal to $\omega(L_l^{-v_k}(v_j)) - \omega(L_t^{v_k}(v_j))$. If there does not exist such a branch, we set $upper_t = \infty$.
5. Initialize lower bound $lb = 0$ and upper bound $ub = 0$. Apply the following algorithm:
For $t = 1$ to r do {
While $lb < upper_t$ do
Set $l = 1$.
Obtain a new sequence $LT^{-v_k}(v_j)$ from $L^{-v_k}(v_j)$ by removing all branches that have common nodes with $L_t^{v_k}(v_j)$. Let rt be the number of branches in sequence $LT^{-v_k}(v_j)$.
While $l \leq rt$ do
While $\omega(L_t^{v_k}(v_j) + lb > l\omega(LT_l^{-v_k}(v_j)) - \Omega_{l-1}(LT^{-v_k}(v_j)) - c_j$ and $l \leq rt$
$l = l + 1$
If $l \leq rt$ then
Set $ub = \omega(LT_l^{-v_k}(v_j)) - \Omega_{l-1}(LT^{-v_k}(v_j)) - \omega(L_t^{v_k}(v_j) - c_j$
If $ub \geq upper_t$ break;
Set $y = \frac{\Omega_{l-1}(LT^{-v_k}(v_j))+\omega(LT_t^{v_k}(v_j))+x}{l}$ for $x \in [lb, ub)$
Set $lb = ub$.
Set $l = l + 1$.
Set $y = \frac{\Omega_{l-1}(LT^{-v_k}(v_j))+\omega(L_t^{v_k}(v_j))+x}{l}$ for $x \in [lb, upper_t)$.
Set $lb = upper_t$.
}

Given a real value x, the corresponding cost for terminal v_k is denoted as $\mathcal{R}_{v_j}^{-1}(x)$. Finally, we give the algorithm to find value $B_k(d_{-k})$.

Algorithm 5.6

Algorithm to find $B_k(d_{-k})$.

1. Remove the terminal v_k and find the multicast tree using the spider structure.
2. For every round i in the first step, we have a graph called G_i and a selected spider with ratio $\rho_i^{-v_k}$. Adding the node v_k and all its incident edges to G_i, we get a graph G_i'.

3. Find the function $y = \mathcal{R}_{v_j}^{-1}(x)$ for every terminal v_j in the graph G_i' using Algorithm 5.5.
4. Calculate $B_k^i(d_{-k}) = \max_{v_j \in V(G_i')}\{\mathcal{R}_{v_j}^{-1}(\rho_i^{-v_k})\}$.
5. $B_k(d_{-k}) = \max_{1 \le i \le r} B_k^i(d_{-k})$.

The correctness of our algorithms is omitted due to space limitations. Notice that for practical implementations, we do not actually have to compute the functions. We are more interested in given some value y, what is the corresponding cost d_k such that the minimum ratio spider centered at the node v_k has a ratio y.

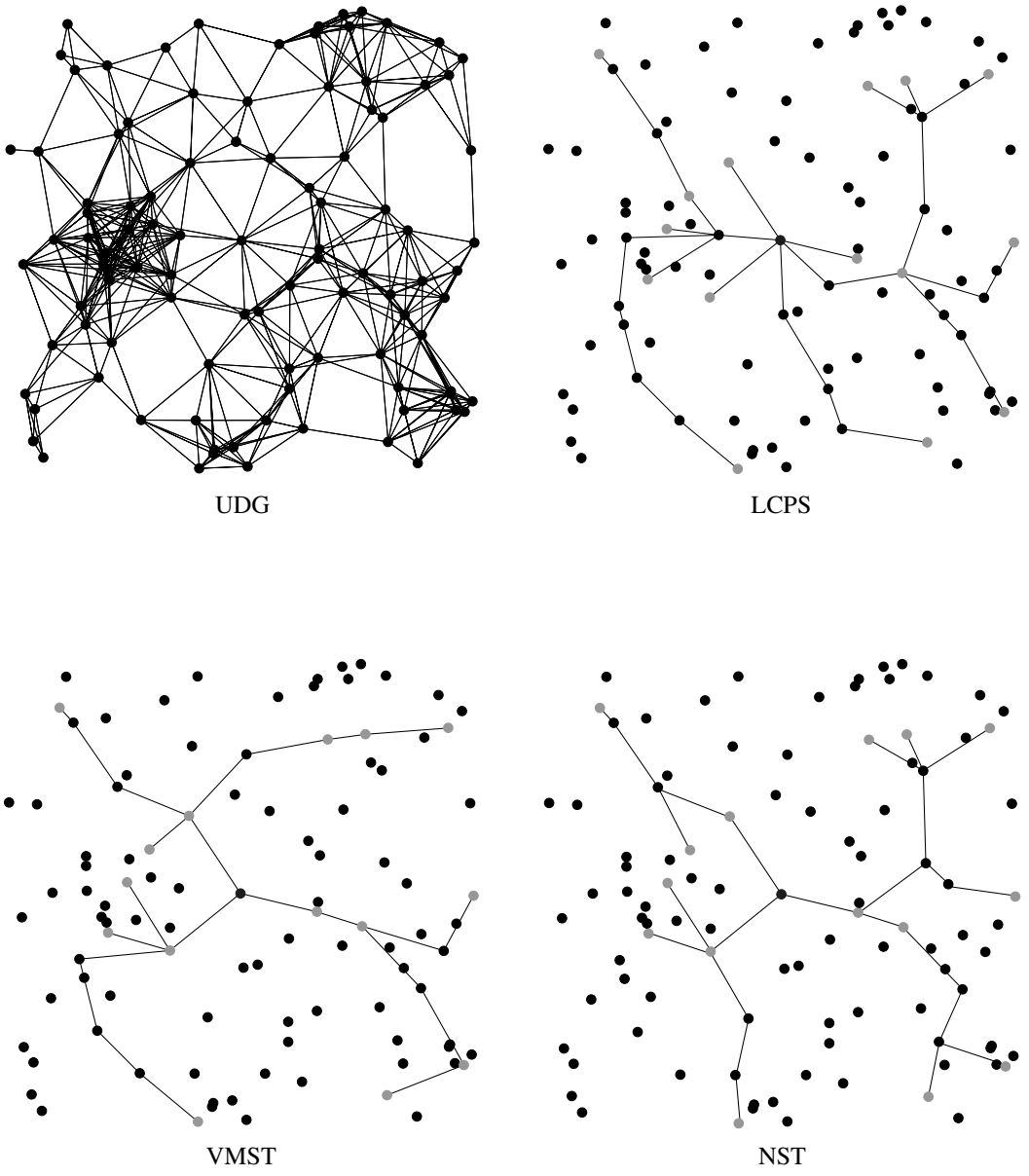
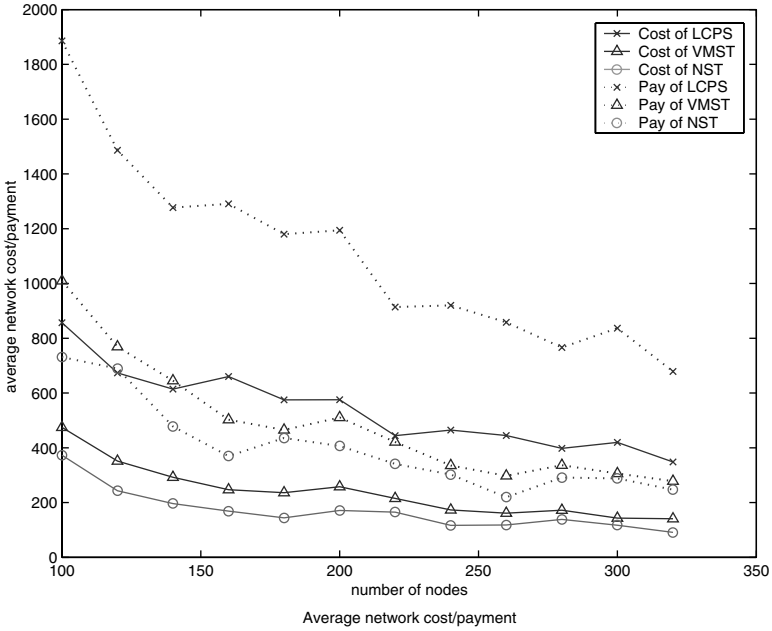


FIGURE 5.4 Multicast structures for node weighted network.



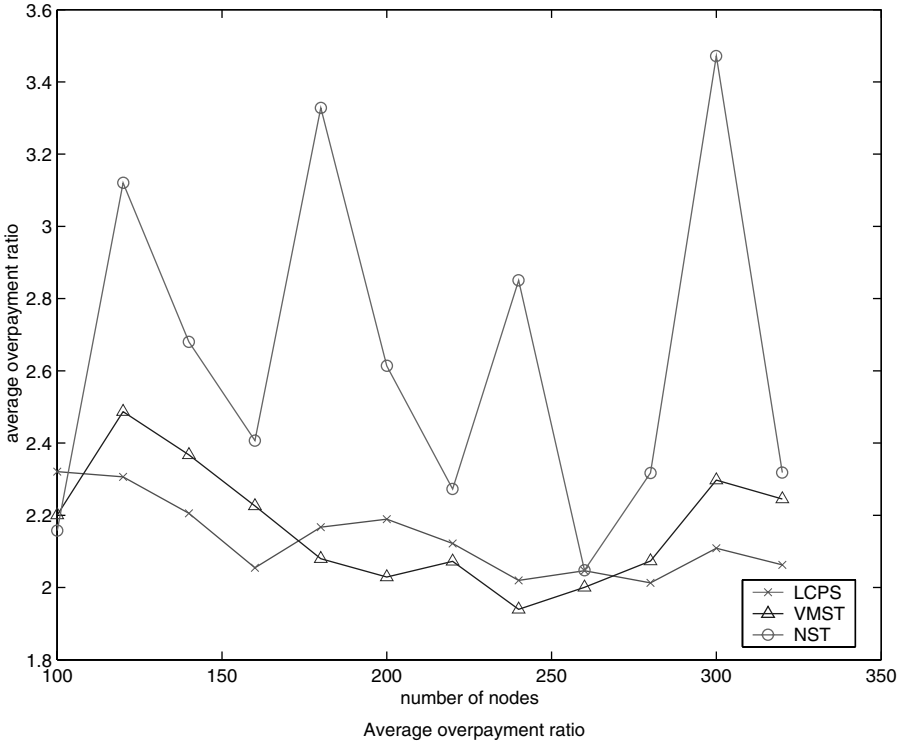


FIGURE 5.5 Results when the number of nodes in the networks are different (from 100 to 320). Here, we fix the transmission range at 300 ft.