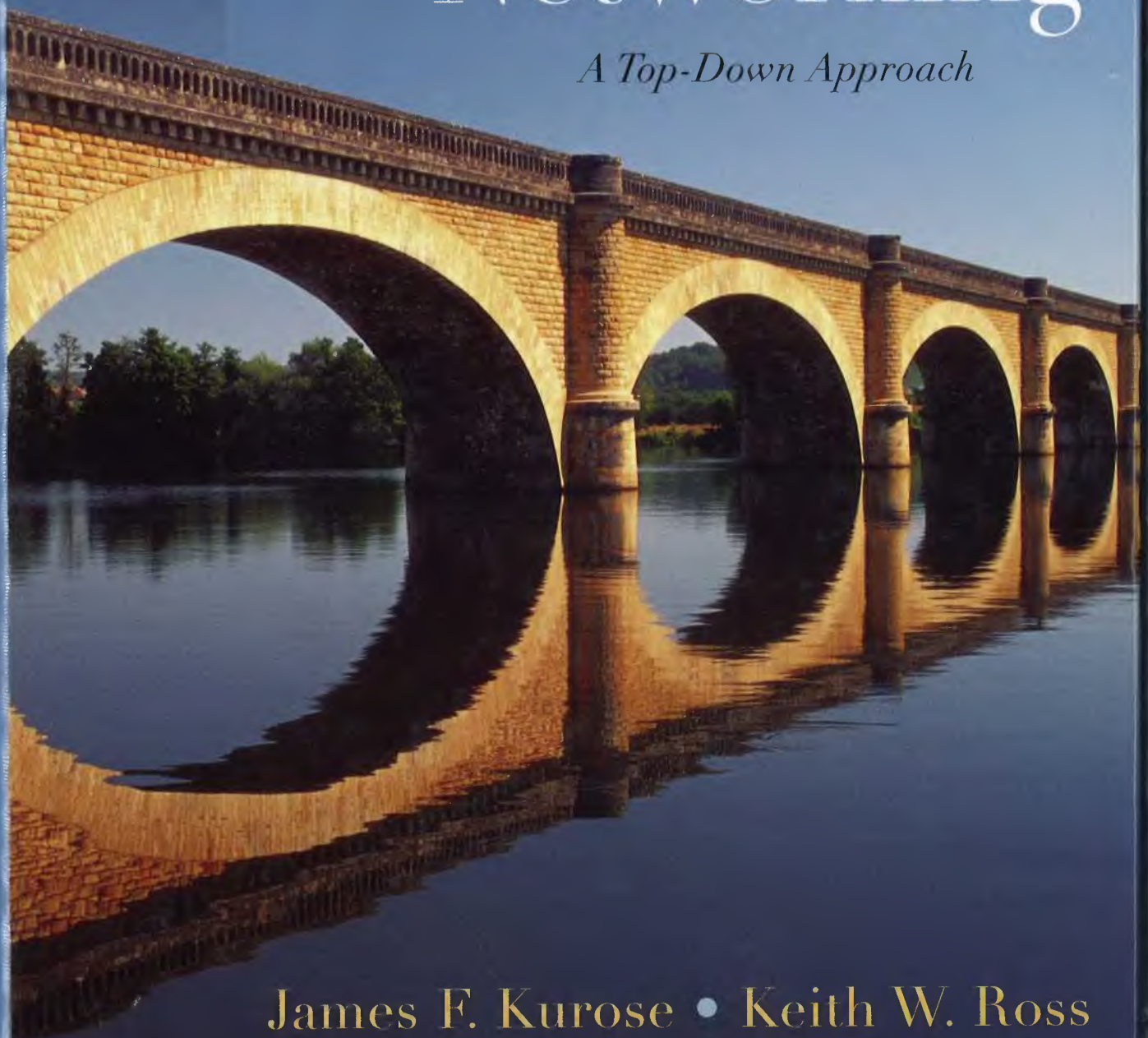
4th Edition

Computer Networking

A Top-Down Approach



James F. Kurose • Keith W. Ross

DELL - Exhibit 1008 - Page 1



CONGRATULATIONS!

Thank you for purchasing a new copy of *Computer Networking: A Top-Down Approach*, Fourth Edition. Your textbook includes six months of prepaid access to the book's Companion Website. This prepaid subscription provides you with full access to all student support areas, including:

- Eleven hands-on Ethereal labs.
- Seven programming labs.
- Java applets illustrating key networking concepts.
- Interactive quizzes to help you assess your basic understanding of the material.
- Links to relevant material.

To Activate Your Prepaid Subscription:

You will need to register online using a computer with an Internet connection and a Web browser. The process takes just a couple of minutes and only needs to be completed once.

1. Go to **http://www.aw-bc.com/kurose_ross** to begin.
2. Click the **Register** button.
3. Use a coin to scratch off the gray coating below and reveal your student access code.* *Do not use a knife or other sharp object which can damage the code.*



*

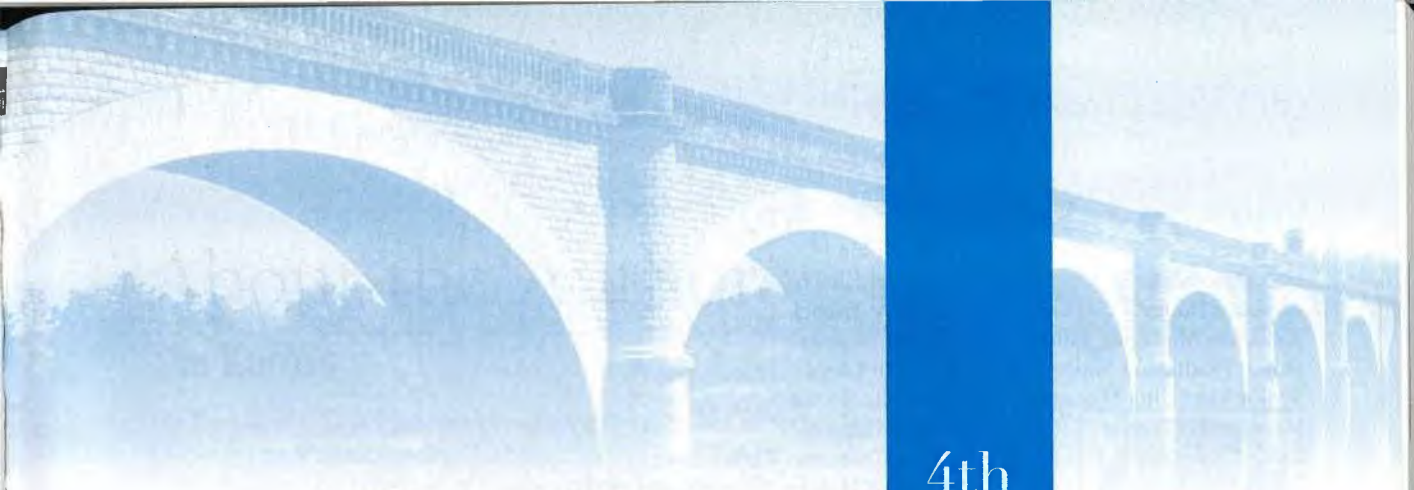
4. On the registration page, enter your student access code. Do not type the dashes. You can use lowercase or uppercase letters.
5. Follow the on-screen instructions. If you need help at any time during the online registration process, simply click the **Need Help?** icon.
6. Once your personal Login Name and Password are confirmed, you can begin using the *Computer Networking: A Top-Down Approach* Companion Website!

To log into this Website after you've registered:

You only need to register for this Companion Website once. After that, you can access the site by going to http://www.aw-bc.com/kurose_ross, clicking "Student Resources," and providing your Login Name and Password when prompted.

*IMPORTANT: The Access Code on this page can only be used once to establish a subscription to the *Computer Networking: A Top-Down Approach*, Fourth Edition Companion Website. This subscription is valid for six months upon activation, and is not transferable. If this access code has already been scratched off, it may no longer be valid. If this is the case, you can purchase a subscription by going to http://www.aw-bc.com/kurose_ross and clicking "Get Access."

DELL - Exhibit 1008 - Page 2



4th Edition

Computer Networking

A Top-Down Approach

James F. Kurose

University of Massachusetts, Amherst

Keith W. Ross

Polytechnic University, Brooklyn



Boston San Francisco New York
London Toronto Sydney Tokyo Singapore Madrid
Mexico City Munich Paris Cape Town Hong Kong Montreal

DELL - Exhibit 1008 - Page 3

Publisher	Greg Tobin
Executive Editor	Michael Hirsch
Assistant Editor	Lindsey Triebel
Associate Managing Editor	Jeffrey Holcomb
Senior Production Supervisor	Marilyn Lloyd
Senior Marketing Manager	Michelle Brown
Marketing Assistant	Sarah Milmore
Cover Designer	Joyce Cosentino Wells
Cover Image	©Robert Harding/Getty Images: Bridge over Dordogne River, Aquitaine, France
Art Director	Janet Theurer/Theurer Briggs Design
Art Studio	Patrice Rossi Calkin/Rossi Illustration & Design
Senior Manufacturing Buyer	Carol Melville
Senior Media Producer	Bethany Tidd
Project Management	Nancy Kotary, Alicia Williams, Scott Harris/Argosy Publishing, Inc.
Composition	Argosy Publishing, Inc.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and Addison-Wesley was aware of a trademark claim, the designations have been printed in initial caps or all caps.

The programs and applications presented in this book have been included for their instructional value. They have been tested with care, but are not guaranteed for any particular purpose. The publisher does not offer any warranties or representations, nor does it accept any liabilities with respect to the programs or applications.

Library of Congress Cataloging-in-Publication Data
Kurose, James F.
Computer networking : a top-down approach / James F. Kurose, Keith W. Ross. — 4th ed.
p. cm.
ISBN 0-321-49770-8
1. Internet. 2. Computer networks. I. Ross, Keith W., 1956- II. Title.
TK5105.875.I57K88 2007
004.6—dc22 2007002094

Copyright © 2008 Pearson Education, Inc.
All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher. Printed in the United States of America. For information on obtaining permission for use of material in this work, please submit a written request to Pearson Education, Inc., Rights and Contracts Department, 75 Arlington Street, Suite 300, Boston, MA 02116, fax your request to 617-848-7047, or e-mail at http://www.pearsoned.com/ legal/permissions.htm.

ISBN-13: 978-0-321-49770-3
ISBN-10: 0-321-49770-8

3 4 5 6 7 8 9 10—CRW—10 09 08

DELL - Exhibit 1008 - Page 4

About the Authors

Jim Kurose



Jim Kurose is a Distinguished University Professor of Computer Science at the University of Massachusetts, Amherst.

Dr. Kurose has received a number of recognitions for his educational activities including Outstanding Teacher Awards from the National Technological University (eight times), the University of Massachusetts, and the Northeast Association of Graduate Schools. He received the IEEE Taylor Booth Education Medal and was recognized for his leadership of Massachusetts' Commonwealth Information Technology Initiative. He has been the recipient of a GE Fellowship, an IBM Faculty Development Award, and a Lilly Teaching Fellowship.

Dr. Kurose is a former Editor-in-Chief of *IEEE Transactions on Communications* and of *IEEE/ACM Transactions on Networking*. He has been active in the program committees for *IEEE Infocom, ACM SIGCOMM, ACM Internet Measurement Conference*, and *ACM SIGMETRICS* for a number of years and has served as Technical Program Co-Chair for those conferences. He is a Fellow of the IEEE and the ACM. His research interests include network protocols and architecture, network measurement, sensor networks, multimedia communication, and modeling and performance evaluation. He holds a PhD in Computer Science from Columbia University.

Keith Ross



Keith Ross is the Leonard J. Shustek Distinguished Chair Professor in Computer Science at Polytechnic University in Brooklyn. From 1985 to 1998 he was a professor in the Department of Systems Engineering at the University of Pennsylvania. From 1998 to 2003 he was a professor in the Multimedia Communications Department at Institute Eurecom in France. Keith Ross is also the principal founder and original CEO of Wimba, which develops Voice over IP and streaming technologies for e-learning markets.

Professor Ross's research interests are in peer-to-peer networking, Internet measurement, video streaming, Web caching, content distribution networks, network security, Voice over IP, and stochastic modeling. He is an IEEE Fellow, and is currently associate editor for *IEEE/ACM Transactions on Networking*. He has served as an advisor to the Federal Trade Commission on P2P file sharing. He has been active in the program committees for *IEEE Infocom, ACM SIGCOMM, ACM Multimedia, ACM Internet Measurement Conference*, and *ACM SIGMETRICS*. He holds a PhD in Computer, Information, and Control Engineering from the University of Michigan.

DELL - Exhibit 1008 - Page 5

To Julie and our three precious
ones—Chris, Charlie, and Nina
JFK

To my wonderful wife, Véronique,
(she's such a great cook among other things!),
and our three daughters, Cécile, Claire, and Katie
KWR

DELL - Exhibit 1008 - Page 6

Preface

Welcome to the fourth edition of *Computer Networking: A Top-Down Approach.* Since the publication of the first edition seven years ago, our book has been adopted for use at many hundreds of colleges and universities, translated into more than 10 languages, and used by over one-hundred thousand students and practitioners worldwide. We've heard from many of these readers and have been overwhelmed by the positive response.

What's New in the Fourth Edition?

We think one important reason for this success has been that our book continues to offer a fresh and timely approach to computer networking instruction. We've made changes in this fourth edition, but we've also kept unchanged what we believe (and the instructors and students who have used our book have confirmed) to be the most important aspects of this book: its top-down approach, its focus on the Internet and a modern treatment of computer networking, its attention to both principles and practice, and its accessible style and approach toward learning about computer networking.

Nevertheless, we've made many important changes in the fourth edition. Given the tremendous importance of network security, we've increased our focus on network security, introducing network security issues in a new section in the very first chapter, adding new security-related material in all chapters, and updating and expanding our coverage of network security in Chapter 8 (which has been dedicated to the topic of network security since our very first edition). We've also updated and expanded our coverage of wireless networks, with additional new material on 802.11 (WiFi), 802.16 (WiMAX), and cellular networks. Our coverage of P2P applications—an increasingly important family of application protocols—includes not only file-sharing protocols, but also file-distribution protocols such as BitTorrent and new peer-to-peer (P2P) multimedia applications such as Voice over IP using Skype. Our chapters on local area networks and multimedia networking have also been streamlined and updated to reflect changes in both the principles and practice in these areas. We've also streamlined Chapter 1 while adding new material on end-to-end throughput analysis. Throughout the book, we've included new state-of-the-art examples and up-to-date references. In the end-of-chapter material, we've added new homework problems, as well as additional hands-on Ethereal labs.

DELL - Exhibit 1008 - Page 7

Audience

This textbook is for a first course on computer networking. It can be used in both computer science and electrical engineering departments. In terms of programming languages, the book assumes only that the student has experience with C, C++, or Java (and even then only in a few places). Although this book is more precise and analytical than many other introductory computer networking texts, it rarely uses any mathematical concepts that are not taught in high school. We have made a deliberate effort to avoid using any advanced calculus, probability, or stochastic process concepts (although we've included some homework problems for students with this advanced background). The book is therefore appropriate for undergraduate courses and for first-year graduate courses. It should also be useful to practitioners in the telecommunications industry.

What Is Unique about This Textbook?

The subject of computer networking is enormously complex, involving many concepts, protocols, and technologies that are woven together in an intricate manner. To cope with this scope and complexity, many computer networking texts are often organized around the "layers" of a network architecture. With a layered organization, students can see through the complexity of computer networking—they learn about the distinct concepts and protocols in one part of the architecture while seeing the big picture of how all parts fit together. From a pedagogical perspective, our personal experience has been that such a layered approach is indeed highly desirable. Nevertheless, we have found the traditional approach of teaching—bottom up, that is, from the physical layer towards the application layer—is not the best approach for a modern course on computer networking.

A Top-Down Approach

Our book broke new ground 7 years ago by treating networking in a top-down manner—that is, by beginning at the application layer and working its way down toward the physical layer. The top-down approach has several important benefits. First, it places emphasis on the application layer (a "high growth area" in networking). Indeed, many of the recent revolutions in computer networking—including the Web, peer-to-peer file sharing, and media streaming—have taken place at the application layer. An early emphasis on application-layer issues differs from the approaches taken in most other texts, which have only a small amount of material on network applications, their requirements, application-layer paradigms (e.g., client-server and peer-to-peer), and application programming interfaces.

Second, our experience as instructors (and that of many instructors who have used this text) has been that teaching networking applications near the beginning of

DELL - Exhibit 1008 - Page 8

the course is a powerful motivational tool. Students are thrilled to learn about how networking applications work—applications such as e-mail and the Web, which most students use on a daily basis. Once a student understands the applications, the student can then understand the network services needed to support these applications. The student can then, in turn, examine the various ways in which such services might be provided and implemented in the lower layers. Covering applications early thus provides motivation for the remainder of the text.

Third, a top-down approach enables instructors to introduce network application development at an early stage. Students not only see how popular applications and protocols work, but also learn how easy it is to create their own network applications and application-level protocols. With the top-down approach, students get early exposure to the notions of application programming interfaces (APIs), service models, and protocols—important concepts that resurface in all subsequent layers. By providing socket programming examples in Java, we highlight the central ideas without confusing students with complex code. Undergraduates in electrical engineering and computer science should not have difficulty following the Java code.

An Internet Focus

With this fourth edition, we have dropped the phrase "Featuring the Internet" from the title, but does this mean that we have dropped our focus on the Internet? Indeed not (and nothing could be farther from the case)! Instead, since the Internet has become so pervasive, we felt that any networking textbook must have a significant focus on the Internet, and thus this phrase was somewhat unnecessary. As in our first three editions, we continue to use the Internet's architecture and protocols as primary vehicles for studying fundamental computer networking concepts. Of course, we also include concepts and protocols from other network architectures. But the spotlight is clearly on the Internet, a fact reflected in our organizing the book around the Internet's five-layer architecture: the application, transport, network, link, and physical layers.

Another benefit of spotlighting the Internet is that most computer science and electrical engineering students are eager to learn about the Internet and its protocols. They know that the Internet is a revolutionary and disruptive technology and can see that it is profoundly changing our world. Given the enormous relevance of the Internet, students are naturally curious about what is "under the hood." Thus, it is easy for an instructor to get students excited about basic principles when using the Internet as the guiding focus.

Addressing Principles

Two of the unique features of the book—its top-down approach and its focus on the Internet—have appeared in the title of the first three editions of our book. If we could have squeezed a *third* phrase into the subtitle, it would have contained the word

DELL - Exhibit 1008 - Page 9

principles. The field of networking is now mature enough that a number of fundamentally important issues can be identified. For example, in the transport layer, the fundamental issues include reliable communication over an unreliable network layer, connection establishment/teardown and handshaking, congestion and flow control, and multiplexing. Two fundamentally important network-layer issues are determining "good" paths between two routers and interconnecting a large number of heterogeneous networks. In the data link layer, a fundamental problem is sharing a multiple access channel. In network security, techniques for providing confidentiality, authentication, and message integrity are all based on cryptographic fundamentals. This text identifies fundamental networking issues and studies approaches towards addressing these issues. The student learning these principles will gain knowledge with a long "shelf life"—long after today's network standards and protocols have become obsolete, the principles they embody will remain important and relevant. We believe that the combination of using the Internet to get the student's foot in the door and then emphasizing fundamental issues and solution approaches will allow the student to quickly understand just about any networking technology.

The Web Site

Purchasing this textbook grants each reader six months of access to a companion Web site for all book readers at http://www.aw.com/kurose-ross, which includes:

- *Interactive learning material.* The Web site contains several interactive Java applets, animating many of the key networking concepts. The site also makes available interactive quizzes that permit students to check their basic understanding of the subject matter. Professors can integrate these interactive features into their lectures or use them as mini labs.
- *Additional technical material.* As we have added new material in each edition of our book, we've had to remove coverage of some existing topics to keep the book at manageable length. For example, to make room for new material on switched LANs, we've removed material on hubs and bridges; to make room for new security material, we've removed material on older security topics (e.g., Kerberos, and key distribution schemes). Material that appeared in earlier editions of the text is still of interest, and can be found on the book's Web site.
- *Programming assignments.* The Web site also provides a number of detailed programming assignments. The programming assignments include building a multithreaded Web server, building an e-mail client with a GUI interface, programming the sender and receiver sides of a reliable data transport protocol, programming a distributed routing algorithm, and more.
- *Ethereal labs.* One's understanding of network protocols can be greatly deepened by seeing them in action. The Web site provides numerous Ethereal assignments that enable students to actually observe the sequence of messages exchanged

DELL - Exhibit 1008 - Page 10

between two protocol entities. The Web site includes separate Ethereal labs on HTTP, DNS, TCP, UDP, IP, ICMP, Ethernet, ARP, WiFi, and SSL.

Pedagogical Features

We have each been teaching computer networking for more than 20 years. Together, we bring more than 45 years of teaching experience to this text, during which time we have taught many thousands of students. We have also been active researchers in computer networking during this time. (In fact, Jim and Keith first met each other as master's students in a computer networking course taught by Mischa Schwartz in 1979 at Columbia University.) We think all this gives us a good perspective on where networking has been and where it is likely to go in the future. Nevertheless, we have resisted temptations to bias the material in this book towards our own pet research projects. We figure you can visit our personal Web sites if you are interested in our research. Thus, this book is about modern computer networking—it is about contemporary protocols and technologies as well as the underlying principles behind these protocols and technologies. We also believe that learning (and teaching!) about networking can be fun. A sense of humor, use of analogies, and real-world examples in this book will hopefully make this material more fun.

Historical Sidebars, Principles in Practice, and a Focus on Security

The field of computer networking has a rich and fascinating history. We have made a special effort in the text to tell the history of computer networking. This is done with a special historical section in Chapter 1 and with about a dozen historical sidebars sprinkled throughout the chapters. In these historical pieces, we cover the invention of packet switching, the evolution of the Internet, the birth of major networking giants such as Cisco and 3Com, and many other important events. Students will be stimulated by these historical pieces. We include special sidebars that highlight important principles in computer networking. These sidebars will help students appreciate some of the fundamental concepts being applied in modern networking. Some of our increased coverage of network security appears in a new series of "Focus on Security" sidebars in each of the core chapters of this book.

Interviews

We have included yet another original feature that should inspire and motivate students—interviews with renowned innovators in the field of networking. We provide interviews with Len Kleinrock, Bram Cohen, Sally Floyd, Vint Cerf, Simon Lam, Charlie Perkins, Henning Schulzrinne, Steven Bellovin, and Jeff Case.

DELL - Exhibit 1008 - Page 11

Supplements for Instructors

We provide a complete supplements package to aid instructors in teaching this course. This material can be accessed from Addison-Wesley's Instructor's Resource Center (http://www.aw.com/irc). Visit the Instructor's Resource Center or send e-mail to computing@aw.com for information about accessing these instructor's supplements.

- *PowerPoint® slides.* We provide PowerPoint slides for all nine chapters. The slides cover each chapter in detail. They use graphics and animations (rather than relying only on monotonous text bullets) to make the slides interesting and visually appealing. We provide the original PowerPoint slides so you can customize them to best suit your own teaching needs. Some of these slides have been contributed by other instructors who have taught from our book.
- *Homework solutions.* We provide a solutions manual for the homework problems in the text, programming assignments, and Ethereal labs.

Chapter Dependencies

The first chapter of this text presents a self-contained overview of computer networking. Introducing many key concepts and terminology, this chapter sets the stage for the rest of the book. All of the other chapters directly depend on this first chapter. We recommend that, after completing Chapter 1, instructors cover Chapters 2 through 5 in sequence, following our top-down philosophy. Each of these five chapters leverages material from the preceding chapters.

After completing the first five chapters, the instructor has quite a bit of flexibility. There are no interdependencies among the last four chapters, so they can be taught in any order. However, each of the last four chapters depends on the material in the first five chapters. Many instructors teach the first five chapters and then teach one of the last four chapters for "dessert."

One Final Note: We'd Love to Hear from You

We encourage instructors and students to create new Java applets that illustrate the concepts and protocols in this book. If you have an applet that you think would be appropriate for this text, please submit it to the authors. If the applet (including notation and terminology) is appropriate, we will be happy to include it on the text's Web site, with an appropriate reference to the authors of the applet. As noted above, we also encourage instructors to send us new homework problems (and solutions) that would complement the current homework problems. We will post these on the instructor-only portion of the Web site.

DELL - Exhibit 1008 - Page 12

We also encourage students and instructors to e-mail us about any comments they might have about our book. It's been wonderful for us to hear from so many instructors and students from around the world about our first three editions. As the saying goes, "Keep those cards and letters coming!" Seriously, please *do* continue to send us interesting URLs, point out typos, disagree with any of our claims, and tell us what works and what doesn't work. Tell us what you think should or shouldn't be included in the next edition. Send your e-mail to kurose@cs.umass.edu and ross@poly.edu.

Acknowledgments

Since we began writing this book in 1996, many people have given us invaluable help and have been influential in shaping our thoughts on how to best organize and teach a networking course. We want to say A BIG THANKS to everyone who has helped us from the earliest first drafts of this book, up to this fourth edition. We are also *very* thankful to the hundreds of readers from around the world—students, faculty, practitioners—who have sent us thoughts and comments on earlier editions of the book and suggestions for future editions of the book. Special thanks go out to:

Al Aho (Columbia University)
Hisham Al-Mubaid (University of Houston-Clear Lake)
Pratima Akkunoor (Arizona State University)
Paul Amer (University of Delaware)
Shamiul Azom (Arizona State University)
Paul Barford (University of Wisconsin)
Bobby Bhattacharjee (University of Maryland)
Steven Bellovin (Columbia University)
Pravin Bhagwat (Wibhu)
Supratik Bhattacharyya (previously at Sprint)
Ernst Biersack (Eurécom Institute)
Shahid Bokhari (University of Engineering & Technology, Lahore)
Jean Bolot (Sprint)
Daniel Brushteyn (former University of Pennsylvania student)
Ken Calvert (University of Kentucky)
Evandro Cantu (Federal University of Santa Catarina)
Jeff Case (SNMP Research International)
Jeff Chaltas (Sprint)
Vinton Cerf (Google)
Byung Kyu Choi (Michigan Technological University)
Bram Cohen (BitTorrent, Inc.)
Constantine Coutras (Pace University)
John Daigle (University of Mississippi)

DELL - Exhibit 1008 - Page 13

Edmundo A. de Souza e Silva (Federal University of Rio de Janiero)
Philippe Decuetos (Eurécom Institute)
Christophe Diot (Thomson Research)
Michalis Faloutsos (University of California at Riverside)
Wu-chi Feng (Oregon Graduate Institute)
Sally Floyd (ICIR, University of California at Berkeley)
Paul Francis (Cornell)
Lixin Gao (University of Massachusetts)
JJ Garcia-Luna-Aceves (University of California at Santa Cruz)
Mario Gerla (University of California at Los Angeles)
David Goodman (Polytechnic University)
Tim Griffin (Cambridge University)
Max Hailperin (Gustavus Adolphus College)
Bruce Harvey (Florida A&M University, Florida State University)
Carl Hauser (Washington State University)
Rachelle Heller (George Washington University)
Phillipp Hoschka (INRIA/W3C)
Wen Hsin (Park University)
Albert Huang (former University of Pennsylvania student)
Esther A. Hughes (Virginia Commonwealth University)
Jobin James (University of California at Riverside)
Sugih Jamin (University of Michigan)
Shivkumar Kalyanaraman (Rensselaer Polytechnic Institute)
Jussi Kangasharju (University of Darmstadt)
Sneha Kasera (University of Utah)
Hyojin Kim (former University of Pennsylvania student)
Leonard Kleinrock (University of California at Los Angeles)
David Kotz (Dartmouth College)
Beshan Kulapala (Arizona State University)
Rakesh Kumar (Polytechnic University)
Miguel A. Labrador (University of South Florida)
Steve Lai (Ohio State University)
Tim-Berners Lee (World Wide Web Consortium)
Lee Leitner (Drexel University)
Brian Levine (University of Massachusetts)
William Liang (former University of Pennsylvania student)
Willis Marti (Texas A&M University)
Nick McKeown (Stanford University)
Josh McKinzie (Park University)
Deep Medhi (University of Missouri, Kansas City)
Bob Metcalfe (International Data Group)
Sue Moon (KAIST)
Erich Nahum (IBM Research)
Christos Papadopoulos (Colorado Sate University)

DELL - Exhibit 1008 - Page 14

Craig Partridge (BBN Technologies)
Radia Perlman (Sun Microsystems)
Jitendra Padhye (Microsoft Research)
Kevin Phillips (Sprint)
George Polyzos (Athens University of Economics and Business)
Sriram Rajagopalan (Arizona State University)
Ramachandran Ramjee (Microsoft Research)
Ken Reek (Rochester Institute of Technology)
Martin Reisslein (Arizona State University)
Jennifer Rexford (Princeton University)
Leon Reznik (Rochester Institute of Technology)
Sumit Roy (University of Washington)
Avi Rubin (Johns Hopkins University)
Dan Rubenstein (Columbia University)
Douglas Salane (John Jay College)
Despina Saparilla (Lucent Bell Labs)
Henning Schulzrinne (Columbia University)
Mischa Schwartz (Columbia University)
Harish Sethu (Drexel University)
K. Sam Shanmugan (University of Kansas)
Prashant Shenoy (University of Massachusetts)
Clay Shields (Georgetown University)
Subin Shrestra (University of Pennsylvania)
Mihail L. Sichitiu (NC State University)
Peter Steenkiste (Carnegie Mellon University)
Tatsuya Suda (University of California at Irvine)
Kin Sun Tam (State University of New York at Albany)
Don Towsley (University of Massachusetts)
David Turner (California State University, San Bernardino)
Nitin Vaidya (University of Illinois)
Michele Weigle (Clemson University)
David Wetherall (University of Washington)
Ira Winston (University of Pennsylvania)
Raj Yavatkar (Intel)
Yechiam Yemini (Columbia University)
Ming Yu (State University of New York at Binghamton)
Ellen Zegura (Georgia Institute of Technology)
Hui Zhang (Carnegie Mellon University)
Lixia Zhang (University of California at Los Angeles)
Shuchun Zhang (former University of Pennsylvania student)
Xiaodong Zhang (Ohio State University)
ZhiLi Zhang (University of Minnesota)
Phil Zimmermann (independent consultant)
Cliff C. Zou (University of Central Florida)

DELL - Exhibit 1008 - Page 15

We also want to thank the entire Addison-Wesley team, who have done an absolutely outstanding job (and who have put up with two very finicky authors!): in particular, Michael Hirsch, Marilyn Lloyd, and Lindsey Triebel. Thanks also to the artists, Janet Theurer and Patrice Rossi Calkin, for their work on the beautiful figures in the second, third and fourth editions of this book, and to Nancy Kotary, Alicia Williams, and Scott Harris for their wonderful production work on this edition. Finally, a most special thanks go to Michael Hirsch, our editor at Addison-Wesley, and Susan Hartman, our former editor at Addison-Wesley. This book would not be what it is (and may well not have been at all) without their graceful management, constant encouragement, nearly infinite patience, good humor, and perseverance.

DELL - Exhibit 1008 - Page 16

Table of Contents

Chapter 1 Computer Networks and the Internet 1

1.1 What Is the Internet? 2
- 1.1.1 A Nuts-and-Bolts Description 2
- 1.1.2 A Services Description 5
- 1.1.3 What Is a Protocol? 7

1.2 The Network Edge 9
- 1.2.1 Client and Server Programs 12
- 1.2.2 Access Networks 12
- 1.2.3 Physical Media 19

1.3 The Network Core 22
- 1.3.1 Circuit Switching and Packet Switching 22
- 1.3.2 How Do Packets Make Their Way Through Packet-Switched Networks? 30
- 1.3.3 ISPs and Internet Backbones 31

1.4 Delay, Loss, and Throughput in Packet-Switched Networks 33
- 1.4.1 Overview of Delay in Packet-Switched Networks 33
- 1.4.2 Queuing Delay and Packet Loss 37
- 1.4.3 End-to-End Delay 40
- 1.4.4 Throughput in Computer Networks 42

1.5 Protocol Layers and Their Service Models 45
- 1.5.1 Layered Architecture 45
- 1.5.2 Messages, Segments, Datagrams, and Frames 51

1.6 Networks Under Attack 53

1.7 History of Computer Networking and the Internet 58
- 1.7.1 The Development of Packet Switching: 1961–1972 58
- 1.7.2 Proprietary Networks and Internetworking: 1972–1980 60
- 1.7.3 A Proliferation of Networks: 1980–1990 62
- 1.7.4 The Internet Explosion: The 1990s 63
- 1.7.5 Recent Developments 64

1.8 Summary 65

Road-Mapping This Book 66

DELL - Exhibit 1008 - Page 17

Homework Problems and Questions 67
Problems 69
Discussion Questions 75
Ethereal Lab 76
Interview: Leonard Kleinrock 78

Chapter 2 Application Layer 81

2.1 Principles of Network Applications 82
2.1.1 Network Application Architectures 82
2.1.2 Processes Communicating 85
2.1.3 Transport Services Available to Applications 88
2.1.4 Transport Services Provided by the Internet 90
2.1.5 Application-Layer Protocols 94
2.1.6 Network Applications Covered in This Book 95
2.2 The Web and HTTP 96
2.2.1 Overview of HTTP 96
2.2.2 Non-persistent and Persistent Connections 98
2.2.3 HTTP Message Format 101
2.2.4 User-Server Interaction: Cookies 106
2.2.5 Web Caching 108
2.2.6 The Conditional GET 112
2.3 File Transfer: FTP 114
2.3.1 FTP Commands and Replies 116
2.4 Electronic Mail in the Internet 116
2.4.1 SMTP 119
2.4.2 Comparison with HTTP 122
2.4.3 Mail Message Formats and MIME 123
2.4.4 Mail Access Protocols 126
2.5 DNS—The Internet's Directory Service 130
2.5.1 Services Provided by DNS 131
2.5.2 Overview of How DNS Works 133
2.5.3 DNS Records and Messages 139
2.6 Peer-to-Peer Applications 144
2.6.1 P2P File Distribution 145
2.6.2 Searching for Information in a P2P Community 151
2.6.3 Case Study: P2P Internet Telephony with Skype 157
2.7 Socket Programming with TCP 159
2.7.1 Socket Programming with TCP 160
2.7.2 An Example Client-Server Application in Java 162
2.8 Socket Programming with UDP 169
2.9 Summary 177

Ch

DELL - Exhibit 1008 - Page 18

Homework Problems and Questions 178
Problems 180
Discussion Questions 187
Socket Programming Assignments 188
Ethereal Labs 190
Interview: Bram Cohen 192

Chapter 3 Transport Layer 195

3.1 Introduction and Transport-Layer Services 196
3.1.1 Relationship Between Transport and Network Layers 196
3.1.2 Overview of the Transport Layer in the Internet 199
3.2 Multiplexing and Demultiplexing 201
3.3 Connectionless Transport: UDP 208
3.3.1 UDP Segment Structure 212
3.3.2 UDP Checksum 212
3.4 Principles of Reliable Data Transfer 214
3.4.1 Building a Reliable Data Transfer Protocol 216
3.4.2 Pipelined Reliable Data Transfer Protocols 225
3.4.3 Go-Back-N (GBN) 228
3.4.4 Selective Repeat (SR) 233
3.5 Connection-Oriented Transport: TCP 240
3.5.1 The TCP Connection 241
3.5.2 TCP Segment Structure 243
3.5.3 Round-Trip Time Estimation and Timeout 248
3.5.4 Reliable Data Transfer 252
3.5.5 Flow Control 260
3.5.6 TCP Connection Management 262
3.6 Principles of Congestion Control 269
3.6.1 The Causes and the Costs of Congestion 270
3.6.2 Approaches to Congestion Control 276
3.6.3 Network-Assisted Congestion-Control Example: ATM ABR Congestion Control 277
3.7 TCP Congestion Control 279
3.7.1 Fairness 287
3.8 Summary 290
Homework Problems and Questions 293
Problems 295
Discussion Questions 304
Programming Assignments 305
Ethereal Labs 305
Interview: Sally Floyd 307

DELL - Exhibit 1008 - Page 19

Chapter 4 The Network Layer **309**

4.1 Introduction 310
4.1.1 Forwarding and Routing 312
4.1.2 Network Service Models 314
4.2 Virtual Circuit and Datagram Networks 317
4.2.1 Virtual-Circuit Networks 318
4.2.2 Datagram Networks 321
4.2.3 Origins of VC and Datagram Networks 323
4.3 What's Inside a Router? 324
4.3.1 Input Ports 326
4.3.2 Switching Fabric 328
4.3.3 Output Ports 331
4.3.4 Where Does Queuing Occur? 331
4.4 The Internet Protocol (IP): Forwarding and Addressing in the Internet 334
4.4.1 Datagram Format 336
4.4.2 IPv4 Addressing 342
4.4.3 Internet Control Message Protocol (ICMP) 357
4.4.4 IPv6 360
4.4.5 A Brief Introduction into IP Security VPNs 366
4.5 Routing Algorithms 368
4.5.1 The Link-State (LS) Routing Algorithm 371
4.5.2 The Distance-Vector (DV) Routing Algorithm 375
4.5.3 Hierarchical Routing 383
4.6 Routing in the Internet 387
4.6.1 Intra-AS Routing in the Internet: RIP 388
4.6.2 Intra-AS Routing in the Internet: OSPF 392
4.6.3 Inter-AS Routing: BGP 395
4.7 Broadcast and Multicast Routing 402
4.7.1 Broadcast Routing Algorithms 403
4.7.2 Multicast 408
4.8 Summary 415
Homework Problems and Questions 416
Problems 419
Discussion Questions 429
Programming Assignment 430
Ethereal Labs 431
Interview: Vinton G. Cerf 432

Chapter 5 The Link Layer and Local Area Networks **435**

5.1 Link Layer: Introduction and Services 437
5.1.1 The Services Provided by the Link Layer 437
5.1.2 Where Is the Link Layer Implemented? 440

DELL - Exhibit 1008 - Page 20

5.2 Error-Detection and -Correction Techniques 442
5.2.1 Parity Checks 444
5.2.2 Checksumming Methods 446
5.2.3 Cyclic Redundancy Check (CRC) 446
5.3 Multiple Access Protocols 449
5.3.1 Channel Partitioning Protocols 451
5.3.2 Random Access Protocols 453
5.3.3 Taking-Turns Protocols 460
5.3.4 Local Area Networks (LANs) 461
5.4 Link-Layer Addressing 463
5.4.1 MAC Addresses 463
5.4.2 Address Resolution Protocol (ARP) 465
5.5 Ethernet 469
5.5.1 Ethernet Frame Structure 471
5.5.2 CSMA/CD: Ethernet's Multiple Access Protocol 475
5.5.3 Ethernet Technologies 477
5.6 Link-Layer Switches 480
5.6.1 Forwarding and Filtering 481
5.6.2 Self-Learning 483
5.6.3 Properties of Link-Layer Switching 484
5.6.4 Switches Versus Routers 485
5.7 PPP: The Point-to-Point Protocol 487
5.7.1 PPP Data Framing 489
5.8 Link Virtualization: A Network as a Link Layer 491
5.8.1 Asynchronous Transfer Mode (ATM) Networks 492
5.8.2 Multiprotocol Label Switching (MPLS) 497
5.9 Summary 500
Homework Problems and Questions 501
Problems 503
Discussion Questions 508
Ethereal Labs 509
Interview: Simon S. Lam 510

Chapter 6 Wireless and Mobile Networks 513
6.1 Introduction 514
6.2 Wireless Links and Network Characteristics 519
6.2.1 CDMA 522
6.3 WiFi: 802.11 Wireless LANs 526
6.3.1 The 802.11 Architecture 527
6.3.2 The 802.11 MAC Protocol 531
6.3.3 The IEEE 802.11 Frame 537
6.3.4 Mobility in the Same IP Subnet 541

DELL - Exhibit 1008 - Page 21

6.3.5 Advanced Features in 802.11 542
6.3.6 Beyond 802.11: Bluetooth and WiMAX 544
6.4 Cellular Internet Access 548
6.4.1 An Overview of Cellular Architecture 548
6.4.2 Cellular Standards and Technologies: A Brief Survey 551
6.5 Mobility Management: Principles 555
6.5.1 Addressing 557
6.5.2 Routing to a Mobile Node 559
6.6 Mobile IP 564
6.7 Managing Mobility in Cellular Networks 570
6.7.1 Routing Calls to a Mobile User 571
6.7.2 Handoffs in GSM 572
6.8 Wireless and Mobility: Impact on Higher-layer Protocols 575
6.9 Summary 578
Homework Problems and Questions 579
Problems 580
Discussion Questions 584
Ethereal Labs 584
Interview: Charlie Perkins 585

Chapter 7 Multimedia Networking 589

7.1 Multimedia Networking Applications 590
7.1.1 Examples of Multimedia Applications 590
7.1.2 Hurdles for Multimedia in Today's Internet 593
7.1.3 How Should the Internet Evolve to Support Multimedia Better? 594
7.1.4 Audio and Video Compression 596
7.2 Streaming Stored Audio and Video 600
7.2.1 Accessing Audio and Video Through a Web Server 600
7.2.2 Sending Multimedia from a Streaming Server to a Helper Application 602
7.2.3 Real-Time Streaming Protocol (RTSP) 604
7.3 Making the Best of the Best-Effort Service 608
7.3.1 The Limitations of a Best-Effort Service 608
7.3.2 Removing Jitter at the Receiver for Audio 611
7.3.3 Recovering from Packet Loss 614
7.3.4 Distributing Multimedia in Today's Internet: Content Distribution Networks 618
7.3.5 Dimensioning Best-Effort Networks to Provide Quality of Service 621
7.4 Protocols for Real-Time Interactive Applications 623
7.4.1 RTP 623
7.4.2 RTP Control Protocol (RTCP) 628

DELL - Exhibit 1008 - Page 22

7.4.3 SIP 631
7.4.4 H.323 637
7.5 Providing Multiple Classes of Service 639
7.5.1 Motivating Scenarios 640
7.5.2 Scheduling and Policing Mechanisms 645
7.5.3 Diffserv 652
7.6 Providing Quality of Service Guarantees 657
7.6.1 A Motivating Example 657
7.6.2 Resource Reservation, Call Admission, Call Setup 659
7.6.3 Guaranteed QoS in the Internet: Intserv and RSVP 661
7.7 Summary 664
Homework Problems and Questions 665
Problems 666
Discussion Questions 673
Programming Assignment 674
Interview: Henning Schulzrinne 676

Chapter 8 Security in Computer Networks 679
8.1 What Is Network Security? 680
8.2 Principles of Cryptography 683
8.2.1 Symmetric Key Cryptography 685
8.2.2 Public Key Encryption 691
8.3 Message Integrity 696
8.3.1 Cryptographic Hash Functions 697
8.3.2 Message Authentication Code 699
8.3.3 Digital Signatures 701
8.4 End-Point Authentication 707
8.4.1 Authentication Protocol *ap1.0* 708
8.4.2 Authentication Protocol *ap2.0* 709
8.4.3 Authentication Protocol *ap3.0* 710
8.4.4 Authentication Protocol *ap3.1* 711
8.4.5 Authentication Protocol *ap4.0* 711
8.4.6 Authentication Protocol *ap5.0* 713
8.5 Securing E-mail 716
8.5.1 Secure E-mail 717
8.5.2 PGP 720
8.6 Securing TCP Connections: SSL 722
8.6.1 The Big Picture 724
8.6.2 A More Complete Picture 727
8.7 Network-Layer Security: IPsec 728
8.7.1 Authentication Header (AH) Protocol 729
8.7.2 The ESP Protocol 731

DELL - Exhibit 1008 - Page 23

8.7.3 SA and Key Management 731
8.8 Securing Wireless LANs 732
8.8.1 Wired Equivalent Privacy (WEP) 732
8.8.2 IEEE802.11i 735
8.9 Operational Security: Firewalls and Intrusion Detection Systems 737
8.9.1 Firewalls 737
8.9.2 Intrusion Detection Systems 744
8.10 Summary 748
Homework Problems and Questions 749
Problems 750
Discussion Questions 753
Ethereal Lab 754
Interview: Steven M. Bellovin 755

Chapter 9 Network Management 757
9.1 What Is Network Management? 758
9.2 The Infrastructure for Network Management 762
9.3 The Internet-Standard Management Framework 766
9.3.1 Structure of Management Information: SMI 768
9.3.2 Management Information Base: MIB 772
9.3.3 SNMP Protocol Operations and Transport Mappings 775
9.3.4 Security and Administration 777
9.4 ASN.1 781
9.5 Conclusion 786
Homework Problems and Questions 787
Problems 787
Discussion Questions 788
Interview: Jeff Case 789

References 791
Index 821

DELL - Exhibit 1008 - Page 24

1
2
2
5
7
7
44
48
49
50
53
54
55

57
58
62
66
68
772
775
777
781
786
787
787
788
789

791
821

Computer Networking

4th Edition

A Top-Down Approach

DELL - Exhibit 1008 - Page 25

CHAPTER 1

Computer Networks and the Internet

From Web browsers in cellular telephones to cafes with public wireless Internet access, from home networks with high-speed broadband access to traditional workplace IT infrastructure with a networked PC on every desk, to networked cars, to networked environmental sensors, to the interplanetary Internet—just as it seems that computer networks are essentially ubiquitous, exciting new applications are developed that extend the reach of today's networks even further. It seems that computer networks are everywhere! This book will provide you with a modern introduction to the dynamic field of computer networking, giving you the principles and practical insights you'll need to understand not only today's networks, but tomorrow's as well.

This first chapter presents an overview of computer networking and the Internet. Our goal here is to paint a broad picture, to see the forest through the trees. We'll cover a lot of ground in this introductory chapter and discuss a lot of pieces of a computer network, without losing sight of the big picture. This chapter lays the groundwork for the rest of the book.

We'll structure our overview of computer networks in this chapter as follows. After introducing some basic terminology and concepts, we'll first examine the basic hardware and software components that make up a network. We'll begin at the network's edge and look at the end systems and network applications running in the network. We'll then explore the core of a computer network, examining the links and the switches that transport data, as well as the access networks and physical

DELL - Exhibit 1008 - Page 26

media that connect end systems to the network core. We'll learn that the Internet is a network of networks, and we'll learn how these networks connect with each other.

After having completed this overview of the edge and core of a computer network, we'll take the broader and more abstract view in the second half of this chapter. We'll examine delay, loss, and throughput in a computer network, and provide simple quantitative models for end-to-end throughput and delay, models that take into account transmission, propagation, and queuing delays. We'll then introduce some of the key architectural principles in computer networking, namely, protocol layering and service models. We'll also learn that computer networks are vulnerable to many different types of attacks; we'll survey some of these attacks and consider how computer networks can be made more secure. Finally, we'll close this chapter with a brief history of computer networking.

1.1 What Is the Internet?

In this book we use the public Internet, a specific computer network, as our principal vehicle for discussing computer networks and their protocols. But what is the Internet? We would like to give a one-sentence definition of the Internet, a definition that you can take home and share with your family and friends. Alas, the Internet is very complex and ever changing, both in terms of its hardware and software components, as well as in the services it provides.

So instead of giving a one-sentence definition, we'll try a more descriptive approach. There are a couple of ways to do this. One way is to describe the nuts and bolts of the Internet, that is, the basic hardware and software components that make up the Internet. Another way is to describe the Internet in terms of a networking infrastructure that provides services to distributed applications. Let's begin with the nuts-and-bolts description, using Figure 1.1 to illustrate our discussion.

1.1.1 A Nuts-and-Bolts Description

The Internet is a computer network that interconnects millions of computing devices throughout the world. Not too long ago, these computing devices were primarily traditional desktop PCs, Linux workstations, and so-called servers that store and transmit information such as Web pages and e-mail messages. Increasingly, however, nontraditional Internet end systems such as personal digital assistants (PDAs), TVs, portable computers, cell phones, Web cams, automobiles, environmental sensing devices, picture frames, and home electrical and security systems are being connected to the Internet. Indeed, the term *computer network* is beginning to sound a bit dated, given the many nontraditional devices that are being hooked up to the Internet. In Internet jargon, all of these devices are called **hosts** or **end systems**. As of July 2006 there were more than 400 million end systems using the Internet, and this number continues to grow rapidly [ISC 2007].

DELL - Exhibit 1008 - Page 27

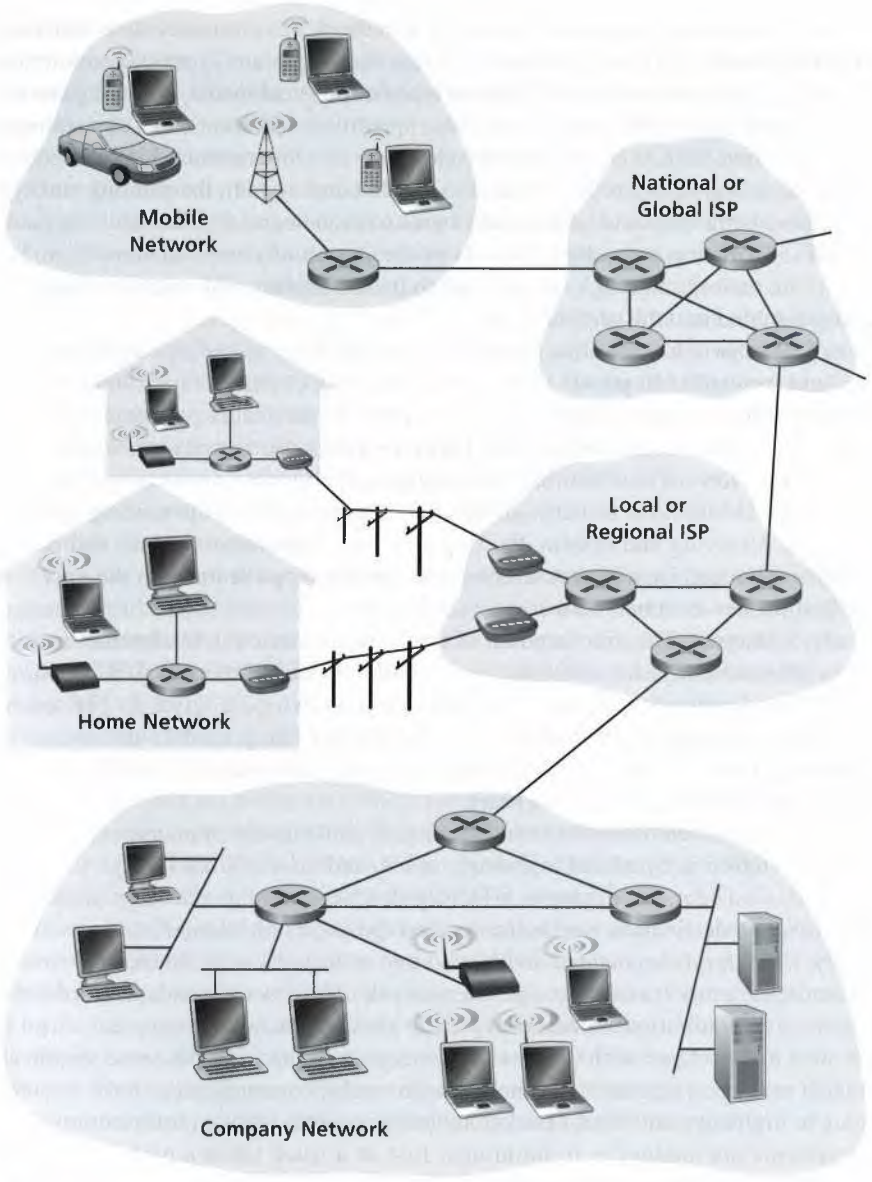


Figure 1.1 ♦ Some pieces of the Internet

DELL - Exhibit 1008 - Page 28

End systems are connected together by a network of **communication links** and **packet switches**. We'll see in Section 1.2 that there are many types of communication links, which are made up of different types of physical media, including coaxial cable, copper wire, fiber optics, and radio spectrum. Different links can transmit data at different rates, with the **transmission rate** of a link measured in bits/second. When one end system has data to send to another end system, the sending end system segments the data and adds header bytes to each segment. The resulting packages of information are called **packets** in the jargon of computer networks. The packets are then sent through the network to the destination end system, where they are reassembled into the original data.

A packet switch takes a packet arriving on one of its incoming communication links and forwards that packet on one of its outgoing communication links. Packet switches come in many shapes and flavors, but the two most prominent types in today's Internet are **routers** and **link-layer switches**. Both types of switches forward packets toward their ultimate destinations. We'll examine routers in detail in Chapter 4 and link-layer switches in detail in Chapter 5. From the sending end system to the receiving end system, the sequence of communication links and packet switches traversed by a packet is known as a **route** or **path** through the network. The first packet-switched networks, created in the 1970s, are the earliest ancestors of today's Internet. The exact amount of traffic being carried in the Internet is difficult to estimate [Odylsko 2003], but one Internet Service Provider (AOL) reported in 2005 that Internet traffic enters its network at a rate of 250 gigabits per second [Gill 2005]. PriMetrica [PriMetrica 2007] estimates that 5 terabits per second of international capacity was used by public carriers in 2006, and that bandwidth usage doubles approximately every two years.

Packet-switched networks (which transport packets) are in many ways similar to transportation networks of highways, roads, and intersections (which transport vehicles). Consider, for example, a factory that needs to move a large amount of cargo to some destination warehouse located thousands of kilometers away. At the factory, the cargo is segmented and loaded into a fleet of trucks. Each of the trucks then independently travels through the network of highways, roads, and intersections to the destination warehouse. At the destination warehouse, the cargo is unloaded and grouped with the rest of the cargo arriving from the same shipment. Thus, in many ways, packets are analogous to trucks, communication links are analogous to highways and roads, packet switches are analogous to intersections, and end systems are analogous to buildings. Just as a truck takes a path through the transportation network, a packet takes a path through a computer network.

End systems access the Internet through **Internet Service Providers (ISPs)**, including residential ISPs such as AOL and local cable or telephone companies; corporate ISPs; university ISPs; and ISPs such as T-Mobile that provide wireless access in airports, hotels, coffee shops, and other public places. Each ISP is in itself a network of packet switches and communication links. ISPs provide a variety of types of network access to the end systems, including 56 kbps dial-up modem access, residential

DELL - Exhibit 1008 - Page 29

broadband access such as cable modem or DSL, high-speed local area network access, and wireless access. ISPs also provide Internet access to content providers, connecting Web sites directly to the Internet. To allow communication among Internet users and to allow users to access worldwide Internet content, these lower-tier ISPs are interconnected through national and international upper-tier ISPs, such as AT&T and Sprint. An upper-tier ISP consists of high-speed routers interconnected with high-speed fiber-optic links. Each ISP network, whether upper-tier or lower-tier, is managed independently, runs the IP protocol (see below), and conforms to certain naming and address conventions. We'll examine ISPs and their interconnection more closely in Section 1.3.

End systems, packet switches, and other pieces of the Internet run **protocols** that control the sending and receiving of information within the Internet. The **Transmission Control Protocol (TCP)** and the **Internet Protocol (IP)** are two of the most important protocols in the Internet. The IP protocol specifies the format of the packets that are sent and received among routers and end systems. The Internet's principal protocols are collectively known as **TCP/IP**. We'll begin looking into protocols in this introductory chapter. But that's just a start—much of this book is concerned with computer network protocols!

Given the importance of protocols to the Internet, it's important that everyone agree on what each and every protocol does. This is where standards come into play. **Internet standards** are developed by the Internet Engineering Task Force (IETF)[IETF 2007]. The IETF standards documents are called **requests for comments (RFCs)**. RFCs started out as general requests for comments (hence the name) to resolve network and protocol design problems that faced the precursor to the Internet. RFCs tend to be quite technical and detailed. They define protocols such as TCP, IP, HTTP (for the Web), and SMTP (for e-mail). There are nearly 5,000 RFCs. Other bodies also specify standards for network components, most notably for network links. The IEEE 802 LAN/MAN Standards Committee [IEEE 802 2007], for example, specifies the Ethernet and wireless WiFi standards.

The public Internet (that is, the global network of networks discussed above) is the network that one typically refers to as *the* Internet. There are also many private networks, such as many corporate and government networks, whose hosts cannot exchange messages with hosts outside of the private network (unless the messages pass through so-called firewalls, which restrict the flow of messages to and from the network). These private networks are often referred to as **intranets**, as they use the same types of hosts, routers, links, and protocols as the public Internet.

1.1.2 A Services Description

The preceding discussion has identified many of the pieces that make up the Internet. But we can also describe the Internet from an entirely different angle—namely, as *an infrastructure that provides services to applications*. These applications include electronic mail, Web surfing, instant messaging, Voice-over-IP (VoIP), Internet radio, video streaming, distributed games, peer-to-peer (P2P) file sharing, tele-

DELL - Exhibit 1008 - Page 30

vision over the Internet, remote login, and much, much more. The applications are said to be **distributed applications** since they involve multiple end systems that exchange data with each other. Importantly, Internet applications run on end systems—they do not run in the packet switches in the network core. As it will become clear as we progress through this book, although packet switches facilitate the exchange of data among end systems, they are not concerned with the application that is the source or sink of data.

Let's explore a little more what we mean by an infrastructure that provides services to applications. To this end, suppose you have an exciting new idea for a distributed Internet application, one that may greatly benefit humanity or one that may simply make you rich and famous. How might you go about transforming this idea into an actual Internet application? Because applications run on end systems, you are going to need to write software pieces that run on the end systems. You might, for example, write your software pieces in Java, C, or C++. Now, because you are developing a distributed Internet application, the software pieces running on the different end systems will need to send data to each other. And here we get to a central issue—one that leads to the alternative way of describing the Internet as a platform for applications. How does one application piece running on one end system instruct the Internet to deliver data to another software piece running on another end system?

End systems attached to the Internet provide an **Application Programming Interface (API)** that specifies how a software piece running on one end system asks the Internet infrastructure to deliver data to a specific destination software piece running on another end system. The Internet API is a set of rules that the sending software piece must follow so that the Internet will deliver the data to the destination software piece. We'll discuss the Internet API in detail in Chapter 2. For now, let's draw upon a simple analogy, one that we will frequently use in this book. Suppose Alice wants to send a letter to Bob using the postal service. Alice, of course, can't just write the letter (the data) and drop the letter out her window. Instead, the postal service requires that Alice put the letter in an envelope; write Bob's full name, address, and zip code in the center of the envelope; seal the envelope; put a stamp in the upper-right-hand corner of the envelope; and finally, drop the envelope into an official postal service mailbox. Thus, the postal service has its own "postal service API," or set of rules, that Alice must follow to have the postal service deliver her letter to Bob. In a similar manner, the Internet has an API that a sending software piece must follow to have the Internet deliver the data to the destination software piece.

The postal service, of course, provides more than one service to its customers. It provides express delivery, reception confirmation, ordinary service, and many more services. In a similar manner, the Internet provides multiple services to its applications. When you develop an Internet application, you too must choose one of the Internet's services for your application. We'll describe the Internet's services in Chapter 2 and describe how the Internet provides those services in Chapter 3.

This second description of the Internet—that is, in terms of an infrastructure for providing services to distributed applications—is an important one. Increasingly,

DELL - Exhibit 1008 - Page 31

advances in the nuts-and-bolts components of the Internet are being driven by the needs of new applications. So it's important to keep in mind that the Internet is an infrastructure in which new applications are being constantly invented and deployed.

We have just given two descriptions of the Internet, one in terms of its hardware and software components, the other in terms of an infrastructure for providing services to distributed applications. But perhaps you are still confused as to what the Internet is. What are packet switching, TCP/IP, and an API? What are routers? What kinds of communication links are present in the Internet? What is a distributed application? How can a toaster or a weather sensor be attached to the Internet? If you feel a bit overwhelmed by all of this now, don't worry—the purpose of this book is to introduce you to both the nuts and bolts of the Internet and the principles that govern how and why it works. We'll explain these important terms and questions in the following sections and chapters.

1.1.3 What Is a Protocol?

Now that we've got a bit of a feel for what the Internet is, let's consider another important buzzword in computer networking: *protocol*. What is a protocol? What does a protocol do? How would you recognize a protocol if you met one?

A Human Analogy

It is probably easiest to understand the notion of a computer network protocol by first considering some human analogies, since we humans execute protocols all of the time. Consider what you do when you want to ask someone for the time of day. A typical exchange is shown in Figure 1.2. Human protocol (or good manners, at least) dictates that one first offer a greeting (the first "Hi" in Figure 1.2) to initiate communication with someone else. The typical response to a "Hi" is a returned "Hi" message. Implicitly, one then takes a cordial "Hi" response as an indication that one can proceed and ask for the time of day. A different response to the initial "Hi" (such as "Don't bother me!" or "I don't speak English," or some unprintable reply) might indicate an unwillingness or inability to communicate. In this case, the human protocol would be not to ask for the time of day. Sometimes one gets no response at all to a question, in which case one typically gives up asking that person for the time. Note that in our human protocol, *there are specific messages we send, and specific actions we take in response to the received reply messages or other events* (such as no reply within some given amount of time). Clearly, transmitted and received messages, and actions taken when these messages are sent or received or other events occur, play a central role in a human protocol. If people run different protocols (for example, if one person has manners but the other does not, or if one understands the concept of time and the other does not) the protocols do not interoperate and no useful work can be accomplished. The same is true in networking—it takes two (or more) communicating entities running the same protocol in order to accomplish a task.

DELL - Exhibit 1008 - Page 32

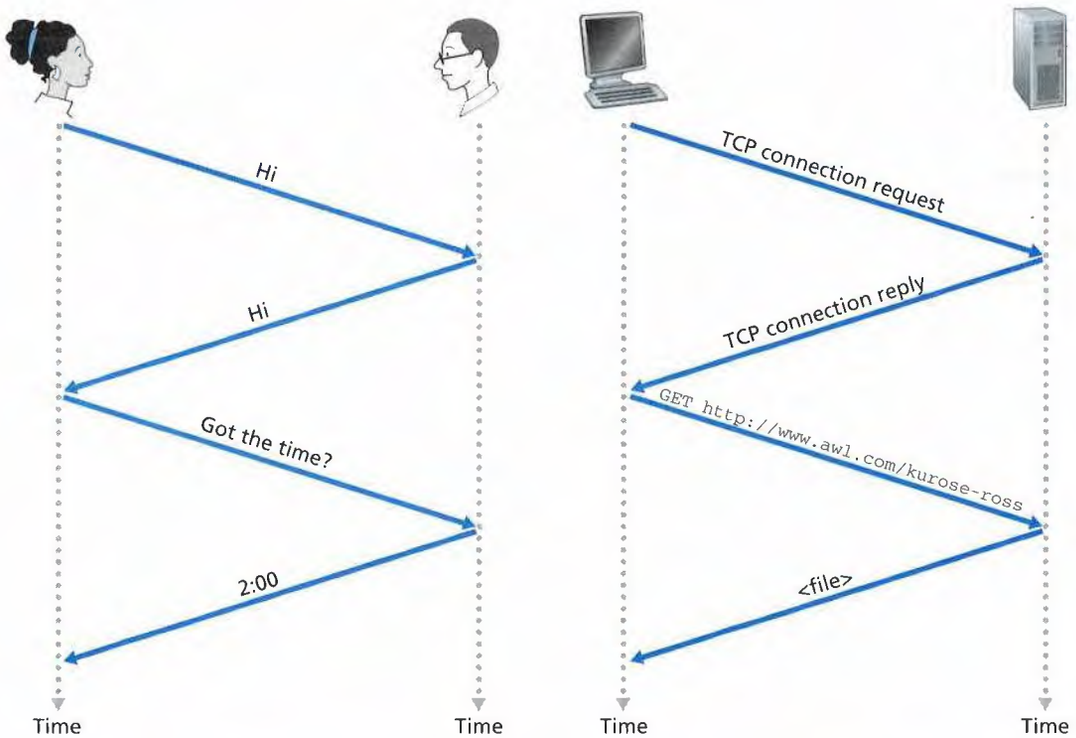


Figure 1.2 ♦ A human protocol and a computer network protocol

Let's consider a second human analogy. Suppose you're in a college class (a computer networking class, for example!). The teacher is droning on about protocols and you're confused. The teacher stops to ask, "Are there any questions?" (a message that is transmitted to, and received by, all students who are not sleeping). You raise your hand (transmitting an implicit message to the teacher). Your teacher acknowledges you with a smile, saying "Yes . . . " (a transmitted message encouraging you to ask your question—teachers *love* to be asked questions), and you then ask your question (that is, transmit your message to your teacher). Your teacher hears your question (receives your question message) and answers (transmits a reply to you). Once again, we see that the transmission and receipt of messages, and a set of conventional actions taken when these messages are sent and received, are at the heart of this question-and-answer protocol.

Network Protocols

A network protocol is similar to a human protocol, except that the entities exchanging messages and taking actions are hardware or software components of some

DELL - Exhibit 1008 - Page 33

device (for example, computer, PDA, cellphone, router, or other network-capable device). All activity in the Internet that involves two or more communicating remote entities is governed by a protocol. For example, hardware-implemented protocols in the network interface cards of two physically connected computers control the flow of bits on the "wire" between the two network interface cards; congestion-control protocols in end systems control the rate at which packets are transmitted between sender and receiver; protocols in routers determine a packet's path from source to destination. Protocols are running everywhere in the Internet, and consequently much of this book is about computer network protocols.

As an example of a computer network protocol with which you are probably familiar, consider what happens when you make a request to a Web server, that is, when you type the URL of a Web page into your Web browser. The scenario is illustrated in the right half of Figure 1.2. First, your computer will send a connection request message to the Web server and wait for a reply. The Web server will eventually receive your connection request message and return a connection reply message. Knowing that it is now OK to request the Web document, your computer then sends the name of the Web page it wants to fetch from that Web server in a GET message. Finally, the Web server returns the Web page (file) to your computer.

Given the human and networking examples above, the exchange of messages and the actions taken when these messages are sent and received are the key defining elements of a protocol:

> *A **protocol** defines the format and the order of messages exchanged between two or more communicating entities, as well as the actions taken on the transmission and/or receipt of a message or other event.*

The Internet, and computer networks in general, make extensive use of protocols. Different protocols are used to accomplish different communication tasks. As you read through this book, you will learn that some protocols are simple and straightforward, while others are complex and intellectually deep. Mastering the field of computer networking is equivalent to understanding the what, why, and how of networking protocols.

1.2 The Network Edge

In the previous section we presented a high-level overview of the Internet and networking protocols. We are now going to delve a bit more deeply into the components of a computer network (and the Internet, in particular). We begin in this section at the edge of a network and look at the components with which we are most familiar—namely, the computers, PDAs, cellphones and other devices that we use on a daily basis. In the next section we'll move from the network edge to the network core and examine switching and routing in computer networks.

DELL - Exhibit 1008 - Page 34

CASE HISTORY

A DIZZYING ARRAY OF INTERNET END SYSTEMS

Not too long ago, the end-system devices connected to the Internet were primarily traditional computers such as desktop machines and powerful servers. Beginning in the late 1990s and continuing today, a wide range of interesting devices of increasing diversity are being connected to the Internet. These devices share the common feature of needing to send and receive digital data to and from other devices. Given the Internet's ubiquity, its well-defined (standardized) protocols, and the availability of Internet-ready commodity hardware, it's natural to use Internet technology to connect these devices together.

Some of these devices seem to have been created purely for fun. A desktop IP-capable picture frame [Ceiva 2007] downloads digital photos from a remote server and displays them in a device that looks like a traditional picture frame; an Internet toaster downloads meteorological information from a server and burns an image of the day's forecast (e.g., mixed clouds and sun) on your morning toast [BBC 2001]. Other devices provide useful information—Web cams display current traffic and weather conditions or monitor a location of interest; Internet-connected home appliances including washing machines, refrigerators, and stoves [Internet Home Alliance 2007] have Web browser interfaces for remote monitoring and control. IP-enabled cell phones put Web browsing, e-mail, and messaging at your fingertips. A new class of networked sensor systems promises to revolutionize how we observe and interact with our environment. Networked sensors that are embedded into the physical environment allow monitoring of buildings, bridges, seismic activity, wildlife habitats, river estuaries, and the lower layers of the atmosphere [CENS 2007, Culler 2004, CASA 2007]. Biomedical sensors may also be embedded and networked [Schwiebert 2001]. Sensed data will be available to remote users in real time. An RFID tag or a tiny embedded sensor affixed to any object can make information about/from that object available on the Internet, leading to an "Internet of things" [ITU 2005].

Recall from the previous section that in computer networking jargon, the computers and other devices connected to the Internet are often referred to as end systems. They are referred to as end systems because they sit at the edge of the Internet, as shown in Figure 1.3. The Internet's end systems include desktop computers (e.g., desktop PCs, Macs, and Linux boxes), servers (e.g., Web and e-mail servers), and mobile computers (e.g., portable computers, PDAs, and phones with wireless Internet connections). Furthermore, an increasing number of alternative devices are being attached to the Internet as end systems (see sidebar).

End systems are also referred to as *hosts* because they host (that is, run) application programs such as a Web browser program, a Web server program, an e-mail reader program, or an e-mail server program. Throughout this book we will use the

DELL - Exhibit 1008 - Page 35

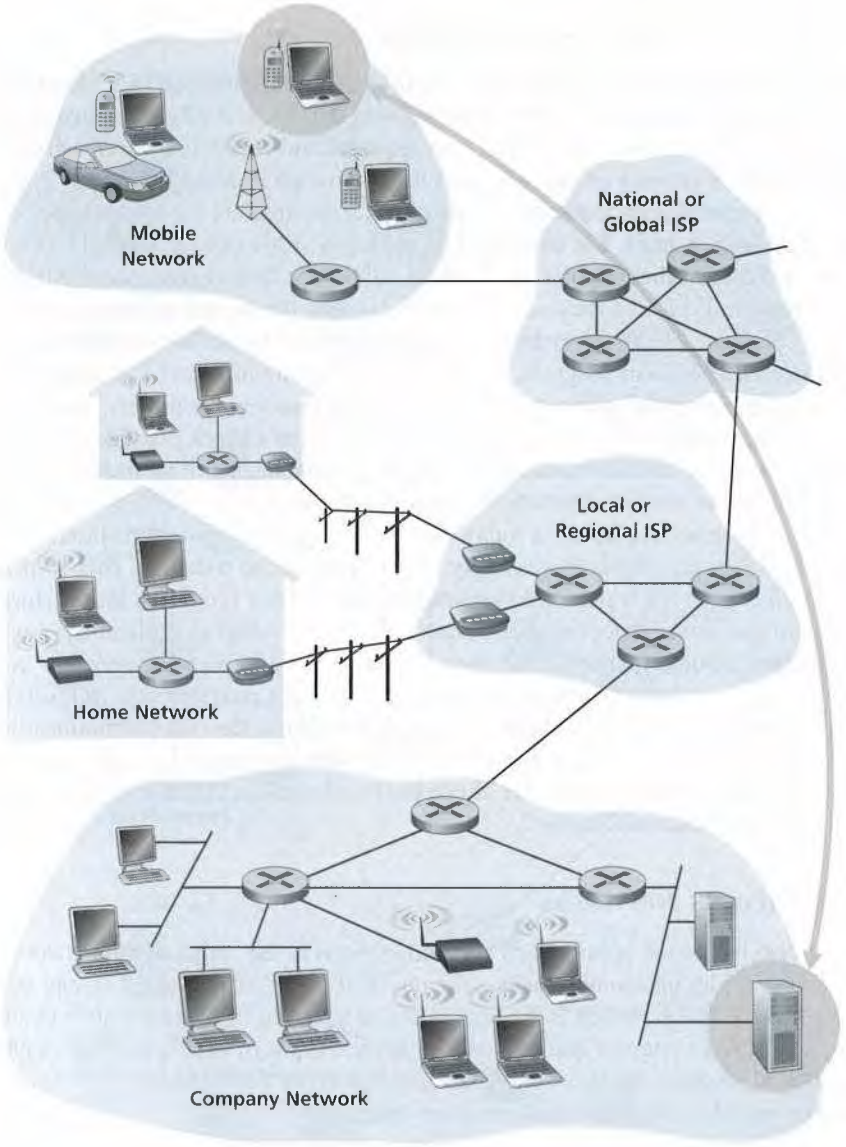


Figure 1.3 ♦ End-system interaction

terms hosts and end systems interchangeably; that is, *host = end system*. Hosts are sometimes further divided into two categories: **clients** and **servers**. Informally, clients tend to be desktop and mobile PCs, PDAs, and so on, whereas servers tend to be more powerful machines that store and distribute Web pages, stream video, relay e-mail, and so on.

DELL - Exhibit 1008 - Page 36

1.2.1 Client and Server Programs

In the context of networking software, there is another definition of a client and server, a definition that we'll refer to throughout this book. A **client program** is a program running on one end system that requests and receives a service from a **server program** running on another end system. Studied in detail in Chapter 2, this client-server model is undoubtedly the most prevalent structure for Internet applications. The Web, e-mail, file transfer, remote login (for example, Telnet), newsgroups, and many other popular applications adopt the client-server model. Since a client program typically runs on one computer and the server program runs on another computer, client-server Internet applications are, by definition, **distributed applications**. The client program and the server program interact by sending each other messages over the Internet. At this level of abstraction, the routers, links, and other nuts and bolts of the Internet serve collectively as a black box that transfers messages between the distributed, communicating components of an Internet application. This is the level of abstraction depicted in Figure 1.3.

Not all Internet applications today consist of pure client programs interacting with pure server programs. Increasingly, many applications today are peer-to-peer (P2P) applications, in which end systems interact and run programs that perform both client and server functions. For example, in P2P file-sharing applications (such as Limewire, eDonkey, and Kazaa), the program in the user's end system acts as a client when it requests a file from another peer; and the program acts as a server when it sends a file to another peer. In Internet telephony, the two communicating parties interact as peers—the communication session is symmetric, with both parties sending and receiving data. We'll compare and contrast client-server and P2P architectures in detail in Chapter 2.

1.2.2 Access Networks

Having considered the applications and end systems at the "edge of the network," let's next consider **access networks**—the physical link(s) that connect an end system to its **edge router,** which is the first router on a path from the end system to any other distant end system. Figure 1.4 shows several types of access links from end system to edge router; the access links are highlighted in thick, shaded lines. Access networks can be loosely classified into three categories:

- *Residential access*, connecting home end systems into the network
- *Company access*, connecting end systems in a business or educational institution into the network
- *Wireless access*, connecting end systems (that are often mobile) into the network

These categories are not hard and fast—for example, some company end systems may use the access technology that we ascribe to residential access, and vice versa. The following descriptions are meant to hold for the common cases.

DELL - Exhibit 1008 - Page 37

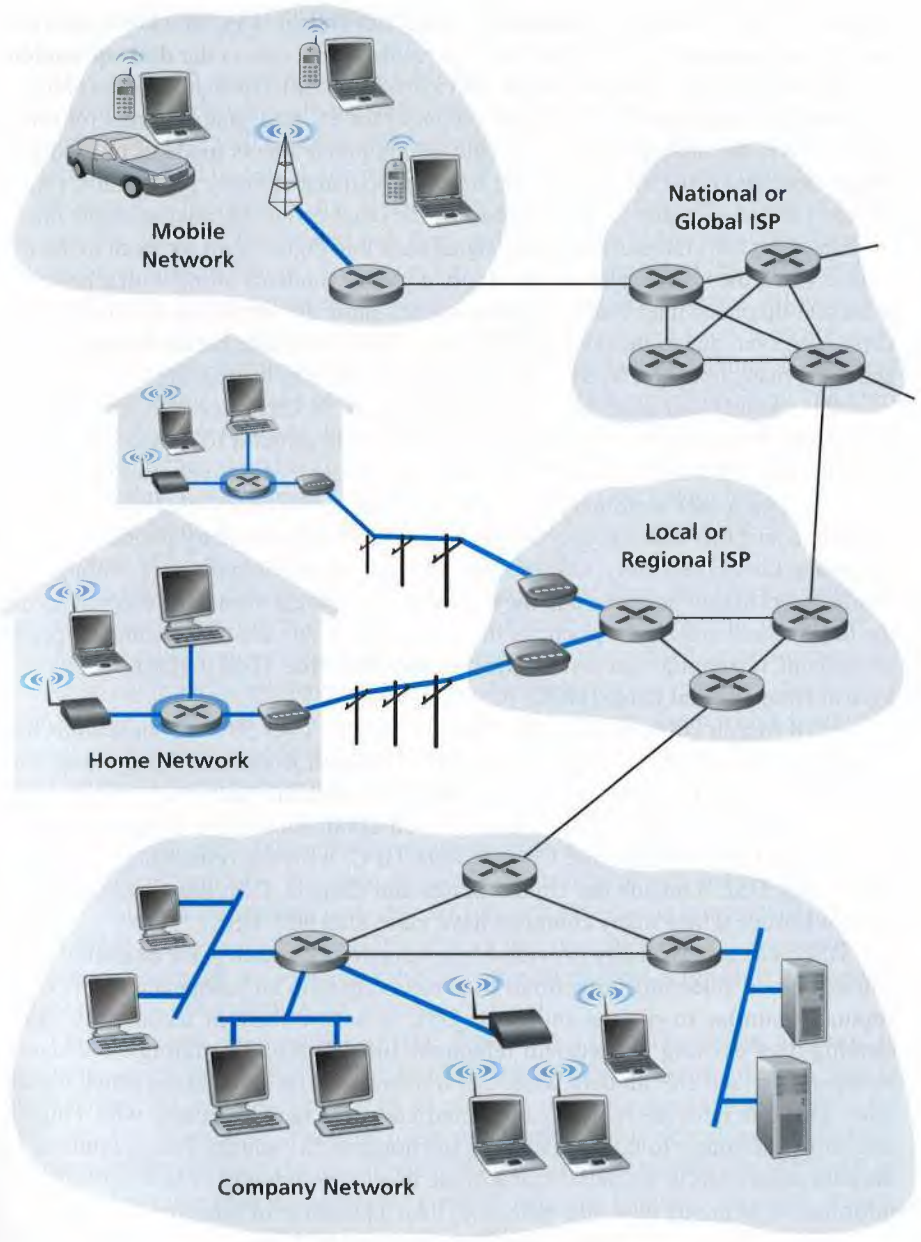


Figure 1.4 ♦ Access networks

DELL - Exhibit 1008 - Page 38

Residential Access

Residential access refers to connecting a home end system (a PC or a home network, see below) to an edge router. One form of residential access is the **dial-up modem** over an ordinary analog telephone line into a residential ISP (such as America Online). The home modem converts the digital output of the PC into analog format for transmission over the analog phone line. This analog phone line is made of twisted-pair copper wire and is the same telephone line used to make ordinary phone calls. (We'll discuss twisted pair later in this section.) At the other end of the analog phone line, a modem in the ISP converts the analog signal back into digital form for input to the ISP router. Thus, the access network is simply a pair of modems along with a point-to-point dial-up phone line. Today's modem speeds allow dial-up access at rates up to 56 kbps. However, due to the poor quality of the twisted-pair line between many homes and ISPs, many users get an effective rate significantly less than 56 kbps.

Many residential users find a dial-up modem's 56 kbps access to be excruciatingly slow. For example, it takes approximately eight minutes to download a single three-minute MP3 song over a 56 kbps dial-up modem. Moreover, dial-up modem access ties up a user's ordinary phone line—while a residential user uses a dial-up modem to surf the Web, the user cannot receive and make ordinary phone calls over the phone line. Fortunately, new broadband access technologies are providing residential users higher bit rates; and they are also providing a means for users to access the Internet and talk on the phone at the same time. There are two common types of broadband residential access: **digital subscriber line (DSL)** [DSL 2007] and **hybrid fiber-coaxial cable (HFC)** [Cable Labs 2007].

As of March 2006, in many developed countries over 50% of households had broadband lines, with rates higher than 80% for South Korea and Hong Kong. The United States and China lead in total number of broadband lines, each with over 40 million lines [Point Topic 2006]. As of March 2006, about 60% of the broadband lines in the United States and Canada were HFC, with the remaining broadband lines being DSL. Outside the United States and Canada, DSL dominates, particularly in Europe where many countries have more than 90% DSL.

DSL access is typically provided by a telephone company (for example, Verizon or France Telecom), sometimes in partnership with an independent ISP. Conceptually similar to dial-up modems, DSL is a new modem technology again running over existing twisted-pair telephone lines. But by restricting the distance between user and ISP modem, DSL can transmit and receive data at much higher rates. The data rates are typically asymmetrical in the two directions, with a higher rate from ISP router to home than from the home to ISP router. The asymmetry in the data rates reflects the belief that a home user is more likely to be a consumer of information (bringing data into the home) than a producer of information.

DSL divides the communication link between the home and the ISP into three separate frequency bands,

- A high-speed downstream channel, in the 50 kHz to 1 MHz band

DELL - Exhibit 1008 - Page 39

- A medium-speed upstream channel, in the 4 kHz to 50 kHz band
- An ordinary two-way telephone channel, in the 0 to 4 kHz band

This approach makes the single DSL link appear as if there were three separate links, so that a telephone call and an Internet connection can share the DSL link at the same time. (We will describe this technique of frequency-division multiplexing in Section 1.3.1). The actual downstream and upstream transmission rate available to the user is a function of the distance between the home modem and the ISP modem, the gauge of the twisted-pair line, and the degree of electrical interference, among other considerations. Engineers have explicitly designed DSL, unlike dial-up modems, for short distances between residential and ISP modems, allowing for substantially higher transmission rates than dial-up access. For users who are close to the ISP modem, different transmission rates are typically available at different prices. For example, Verizon offers transmission rates that vary from about 768 kbps to 1.5 Mbps for downstream, and 128 kbps to 768 kbps for upstream. There are also a variety of higher-speed DSL technologies enjoying penetration in a handful of countries today. For example, Very-high speed DSL (VDSL), with highest penetration today in South Korea and Japan, provides impressive rates of 12–55 Mbps for downstream and 1.6–20 Mbps for upstream [DSL 2007].

While DSL and dial-up modems use ordinary phone lines, HFC access networks are extensions of the current cable network used for broadcasting cable television. In a traditional cable system, a cable head end broadcasts through a distribution network of coaxial cable and amplifiers to residences. As illustrated in Figure 1.5, fiber optics connects the cable head end to neighborhood-level junctions, from which traditional coaxial cable is then used to reach individual houses and apartments. Each neighborhood junction typically supports 500 to 5,000 homes.

As with DSL, HFC requires special modems, called **cable modems**. Companies that provide cable Internet access require their customers to either purchase or lease a modem. Typically, the cable modem is an external device and connects to the home PC through an Ethernet port. (We will discuss Ethernet in great detail in Chapter 5.) Cable modems divide the HFC network into two channels, a downstream and an upstream channel. As with DSL, the downstream channel is typically allocated a higher transmission rate than the upstream channel.

One important characteristic of HFC is that it is a shared broadcast medium. In particular, every packet sent by the head end travels downstream on every link to every home; and every packet sent by a home travels on the upstream channel to the head end. For this reason, if several users are simultaneously downloading MP3s on the downstream channel, the actual rate at which each user receives its MP3 will be significantly lower than the aggregate downstream rate. On the other hand, if there are only a few active users and they are all Web surfing, then each of the users may actually receive Web pages at the full downstream rate, because users will rarely request a Web page at exactly the same time. Because the upstream channel is also shared, a distributed multiple-access protocol is needed to coordinate transmissions

DELL - Exhibit 1008 - Page 40

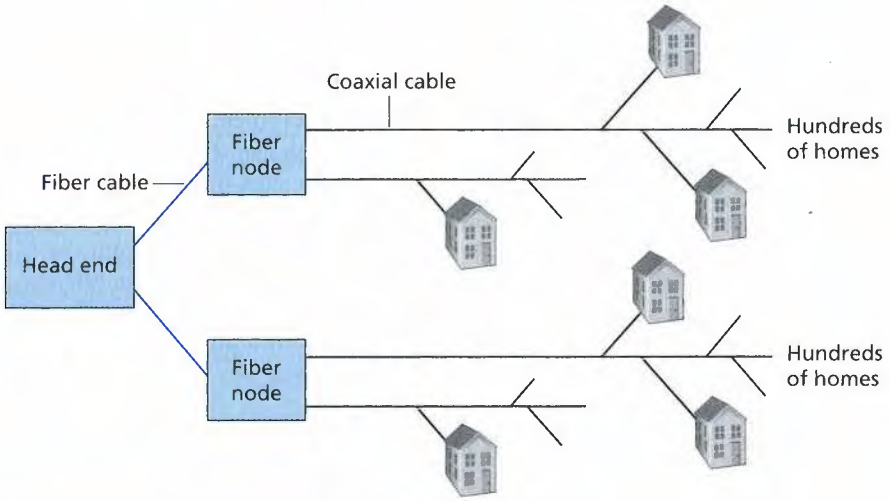


Figure 1.5 ♦ A hybrid fiber-coaxial access network

and avoid collisions. (We'll discuss this collision issue in some detail when we discuss Ethernet in Chapter 5.) Advocates of DSL are quick to point out that DSL is a point-to-point connection between the home and ISP, and therefore the entire transmission capacity of the DSL link between the home and the ISP is dedicated rather than shared. Cable advocates, however, argue that a reasonably dimensioned HFC network provides higher transmission rates than DSL. The battle between DSL and HFC for high-speed residential access is raging, particularly in North America. In rural areas, where neither DSL nor HFC is available, a satellite link can be used to connect a residence to the Internet at speeds of more than 1 Mbps; StarBand and HughesNet are two such satellite access providers.

One of the attractive features of DSL, HFC, and satellite access is that the services are **always on**; that is, the user can leave his or her computer on and remain permanently connected to an ISP while simultaneously making and receiving ordinary telephone calls.

Company Access

On corporate and university campuses, a local area network (LAN) is typically used to connect an end system to the edge router. As we will see in Chapter 5, there are many types of LAN technology. However, Ethernet technology is currently by far the most prevalent access technology in company networks. Today's Ethernet technology operates at 100 Mbps or 1 Gbps (or even 10 Gbps). It uses either twisted-pair copper wire or coaxial cable to connect a number of end systems with each other and with an edge router. The edge router is responsible for routing

DELL - Exhibit 1008 - Page 41

packets that have destinations outside of that LAN. Like HFC, Ethernet can use a shared medium, so that end users share the transmission rate of the LAN. More recently, shared Ethernet technology has migrated to switched Ethernet technology. Switched Ethernet uses a star topology with hosts connected directly to a "switch", allowing all the hosts to simultaneously send and receive at the full transmission rate of the LAN. We will explore shared- and switched-Ethernet in detail in Chapter 5.

Wireless Access

Accompanying the current Internet revolution, the wireless revolution is also having a profound impact on the way people work and live. Today, more people in Europe have a mobile phone than a PC or a car. And the wireless trend is continuing, with many analysts predicting that wireless (and often mobile) handheld devices—such as mobile phones and PDAs—will overtake wired computers as the dominant Internet access devices throughout the world. Today, there are two common types of wireless Internet access. In a **wireless LAN**, wireless users transmit/receive packets to/from a base station (also known as a wireless access point) within a radius of a few tens of meters. The base station is typically connected to the wired Internet and thus serves to connect wireless users to the wired network. In **wide-area wireless access networks**, packets are transmitted over the same wireless infrastructure used for cellular telephony, with the base station thus being managed by a telecommunications provider. This provides wireless access to users within a radius of tens of kilometers of the base station.

Wireless LANs, based on IEEE 802.11 technology (also known as wireless Ethernet and WiFi), are currently enjoying widespread deployment in university departments, business offices, cafes, and homes. Many universities install IEEE 802.11 base stations across their campuses, allowing students to send and receive e-mail or surf the Web from anywhere on campus (for example, library, dorm room, classroom, or outdoor campus bench). In many cities, one can stand on a street corner and be within range of ten or twenty base stations (for a browseable global map of 802.11 base stations that have been discovered and logged on a Web site by people who take great enjoyment in doing such things, see [wiggle.net 2007]). The most commonly deployed 802.11 technology, which we'll discuss in detail in Chapter 6, provides a shared transmission rate of 54 Mbps.

Today many homes are combining broadband residential access (that is, cable modems or DSL) with inexpensive wireless LAN technology to create powerful home networks. Figure 1.6 shows a schematic of a typical home network. This home network consists of a roaming laptop as well as a wired PC; a base station (the wireless access point), which communicates with the wireless PC; a cable modem, providing broadband access to the Internet; and a router, which interconnects the base station and the stationary PC with the cable modem. This network allows household members to have broadband access to the Internet, with one member roaming from

DELL - Exhibit 1008 - Page 42

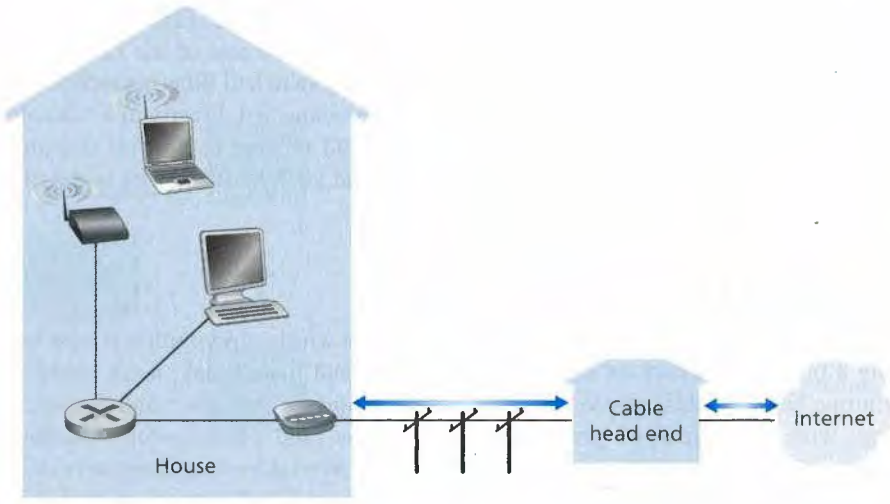


Figure 1.6 ♦ A schematic of a typical home network

the kitchen to the backyard to the bedrooms. The total fixed cost for such a network is less than $150 (including the cable/DSL modem).

When you access the Internet through wireless LAN technology, you typically need to be within a few tens of meters of a base station. This is feasible for home access, coffee shop access, and, more generally, access within and around a building. But what if you are on the beach or in your car and you need Internet access? For such wide-area access, roaming Internet users make use of the cellular phone infrastructure, accessing base stations that are up to tens of kilometers away. Conceptually, this is similar to a home user with a dial-up connection to the Internet over a wired telephone line, except that now the cellular telephony infrastructure, rather than the wired telephony infrastructure, is used.

Telecommunications companies have made enormous investments in so-called Third Generation (3G) wireless, which provides packet-switched wide-area wireless Internet access at speeds in excess of 1 Mbps. Two important standards for wide-area wireless Internet are EVDO (Evolution-Data Optimized—arguably one of the worst acronyms ever invented!) and HSDPA (High-Speed Downlink Packet Access), with many cellular telephone network operators offering (or planning to offer) one or the other of these two standards. As always, there is a potential "killer" technology waiting to dethrone these standards. WiMAX [Intel WiMAX 2007, WiMAX Forum 2007], also known as IEEE 802.16, is a long-distance cousin of the 802.11 WiFi protocol discussed above. WiMAX operates independently of the cellular network and promises speeds of 5 to 10 Mbps or higher over distances of tens of kilometers. Although there are no widespread WiMAX deployments as of late 2006, Sprint-Nextel has committed billions of dollars towards deploying WiMAX in 2007 and beyond. We'll cover WiFi, WiMAX, and 3G in detail in Chapter 6.

DELL - Exhibit 1008 - Page 43

1.2.3 Physical Media

In the previous subsection, we gave an overview of some of the most important network access technologies in the Internet. As we described these technologies, we also indicated the physical media used. For example, we said that HFC uses a combination of fiber cable and coaxial cable. We said that dial-up 56 kbps modems and DSL use twisted-pair copper wire. And we said that mobile access networks use the radio spectrum. In this subsection we provide a brief overview of these and other transmission media that are commonly used in the Internet.

In order to define what is meant by a physical medium, let us reflect on the brief life of a bit. Consider a bit traveling from one end system, through a series of links and routers, to another end system. This poor bit gets kicked around and transmitted many, many times! The source end system first transmits the bit, and shortly thereafter the first router in the series receives the bit; the first router then transmits the bit, and shortly thereafter the second router receives the bit; and so on. Thus our bit, when traveling from source to destination, passes through a series of transmitter-receiver pairs. For each transmitter-receiver pair, the bit is sent by propagating electromagnetic waves or optical pulses across a **physical medium**. The physical medium can take many shapes and forms and does not have to be of the same type for each transmitter-receiver pair along the path. Examples of physical media include twisted-pair copper wire, coaxial cable, multimode fiber-optic cable, terrestrial radio spectrum, and satellite radio spectrum. Physical media fall into two categories: **guided media** and **unguided media**. With guided media, the waves are guided along a solid medium, such as a fiber-optic cable, a twisted-pair copper wire, or a coaxial cable. With unguided media, the waves propagate in the atmosphere and in outer space, such as in a wireless LAN or a digital satellite channel.

But before we get into the characteristics of the various media types, let us say a few words about their costs. The actual cost of the physical link (copper wire, fiber-optic cable, and so on) is often relatively minor compared with other networking costs. In particular, the labor cost associated with the installation of the physical link can be orders of magnitude higher than the cost of the material. For this reason, many builders install twisted pair, optical fiber, and coaxial cable in every room in a building. Even if only one medium is initially used, there is a good chance that another medium could be used in the near future, and so money is saved by not having to lay additional wires in the future.

Twisted-Pair Copper Wire

The least expensive and most commonly used guided transmission medium is twisted-pair copper wire. For over a hundred years it has been used by telephone networks. In fact, more than 99 percent of the wired connections from the telephone handset to the local telephone switch use twisted-pair copper wire. Most of us have seen twisted pair in our homes and work environments. Twisted pair consists of two insulated copper wires, each about 1 mm thick, arranged in a regular spiral pattern.

DELL - Exhibit 1008 - Page 44

The wires are twisted together to reduce the electrical interference from similar pairs close by. Typically, a number of pairs are bundled together in a cable by wrapping the pairs in a protective shield. A wire pair constitutes a single communication link. **Unshielded twisted pair (UTP)** is commonly used for computer networks within a building, that is, for LANs. Data rates for LANs using twisted pair today range from 10 Mbps to 1 Gbps. The data rates that can be achieved depend on the thickness of the wire and the distance between transmitter and receiver.

When fiber-optic technology emerged in the 1980s, many people disparaged twisted pair because of its relatively low bit rates. Some people even felt that fiber-optic technology would completely replace twisted pair. But twisted pair did not give up so easily. Modern twisted-pair technology, such as category 5 UTP, can achieve data rates of 1 Gbps for distances up to a hundred meters. In the end, twisted pair has emerged as the dominant solution for high-speed LAN networking.

As discussed earlier, twisted pair is also commonly used for residential Internet access. We saw that dial-up modem technology enables access at rates of up to 56 kbps over twisted pair. We also saw that DSL (digital subscriber line) technology has enabled residential users to access the Internet at rates in excess of 6 Mbps over twisted pair (when users live close to the ISP's modem).

Coaxial Cable

Like twisted pair, coaxial cable consists of two copper conductors, but the two conductors are concentric rather than parallel. With this construction and special insulation and shielding, coaxial cable can have high bit rates. Coaxial cable is quite common in cable television systems. As we saw earlier, cable television systems have recently been coupled with cable modems to provide residential users with Internet access at rates of 1 Mbps or higher. In cable television and cable Internet access, the transmitter shifts the digital signal to a specific frequency band, and the resulting analog signal is sent from the transmitter to one or more receivers. Coaxial cable can be used as a guided **shared medium**. Specifically, a number of end systems can be connected directly to the cable, with each of the end systems receiving whatever is sent by the other end systems.

Fiber Optics

An optical fiber is a thin, flexible medium that conducts pulses of light, with each pulse representing a bit. A single optical fiber can support tremendous bit rates, up to tens or even hundreds of gigabits per second. They are immune to electromagnetic interference, have very low signal attenuation up to 100 kilometers, and are very hard to tap. These characteristics have made fiber optics the preferred long-haul guided transmission media, particularly for overseas links. Many of the long-distance telephone networks in the United States and elsewhere now use fiber optics exclusively. Fiber optics is also prevalent in the backbone of the Internet. However,

DELL - Exhibit 1008 - Page 45

the high cost of optical devices—such as transmitters, receivers, and switches—has hindered their deployment for short-haul transport, such as in a LAN or into the home in a residential access network. The Optical Carrier (OC) standard link speeds range from 51.8 Mbps to 39.8 Gbps; these specifications are often referred to as OC-n, where the link speed equals $n \times 51.8$ Mbps. Standards in use today include OC-1, OC-3, OC-12, OC-24, OC-448, OC-96, OC-192, OC-768. [IEC Optical 2007; Goralski 2001; Ramaswami 1998; and Mukherjee 1997] provide coverage of various aspects of optical networking.

Terrestrial Radio Channels

Radio channels carry signals in the electromagnetic spectrum. They are an attractive medium because they require no physical wire to be installed, can penetrate walls, provide connectivity to a mobile user, and can potentially carry a signal for long distances. The characteristics of a radio channel depend significantly on the propagation environment and the distance over which a signal is to be carried. Environmental considerations determine path loss and shadow fading (which decrease the signal strength as the signal travels over a distance and around/through obstructing objects), multipath fading (due to signal reflection off of interfering objects), and interference (due to other transmissions and electromagnetic signals).

Terrestrial radio channels can be broadly classified into two groups: those that operate in local areas, typically spanning from ten to a few hundred meters; and those that operate in the wide area, spanning tens of kilometers. The wireless LAN technologies described in Section 1.2.2 use local-area radio channels; the cellular access technologies use wide-area radio channels. We'll discuss radio channels in detail in Chapter 6.

Satellite Radio Channels

A communication satellite links two or more Earth-based microwave transmitter/receivers, known as ground stations. The satellite receives transmissions on one frequency band, regenerates the signal using a repeater (discussed below), and transmits the signal on another frequency. Two types of satellites are used in communications: **geostationary satellites** and **low-earth orbiting (LEO) satellites**.

Geostationary satellites permanently remain above the same spot on Earth. This stationary presence is achieved by placing the satellite in orbit at 36,000 kilometers above Earth's surface. This huge distance from ground station through satellite back to ground station introduces a substantial signal propagation delay of 280 milliseconds. Nevertheless, satellite links, which can operate at speeds of hundreds of Mbps, are often used in telephone networks and in the backbone of the Internet. As discussed in Section 1.2.2, satellite links are also increasingly being used for high-speed residential Internet access in areas without access to DSL or cable-based Internet access.

DELL - Exhibit 1008 - Page 46

LEO satellites are placed much closer to Earth and do not remain permanently above one spot on Earth. They rotate around Earth (just as the Moon does) and may communicate with each other, as well as with ground stations. To provide continuous coverage to an area, many satellites need to be placed in orbit. There are currently many low-altitude communication systems in development. Lloyd's satellite constellations Web page [Wood 2007] provides and collects information on satellite constellation systems for communications. LEO satellite technology may be used for Internet access sometime in the future.

1.3 The Network Core

Having examined the Internet's edge, let us now delve more deeply inside the network core—the mesh of packet switches and links that interconnects the Internet's end systems. Figure 1.7 highlights the network core with thick, shaded lines.

1.3.1 Circuit Switching and Packet Switching

There are two fundamental approaches to moving data through a network of links and switches: **circuit switching** and **packet switching**. In circuit-switched networks, the resources needed along a path (buffers, link transmission rate) to provide for communication between the end systems are *reserved* for the duration of the communication session between the end-systems. In packet-switched networks, these resources are *not* reserved; a session's messages use the resources on demand, and as a consequence, may have to wait (that is, queue) for access to a communication link. As a simple analogy, consider two restaurants, one that requires reservations and another that neither requires reservations nor accepts them. For the restaurant that requires reservations, we have to go through the hassle of calling before we leave home. But when we arrive at the restaurant we can, in principle, immediately communicate with the waiter and order our meal. For the restaurant that does not require reservations, we don't need to bother to reserve a table. But when we arrive at the restaurant, we may have to wait for a table before we can communicate with the waiter.

The ubiquitous telephone networks are examples of circuit-switched networks. Consider what happens when one person wants to send information (voice or facsimile) to another over a telephone network. Before the sender can send the information, the network must establish a connection between the sender and the receiver. This is a *bona fide* connection for which the switches on the path between the sender and receiver maintain connection state for that connection. In the jargon of telephony, this connection is called a **circuit**. When the network establishes the circuit, it also reserves a constant transmission rate in the network's links for the

DELL - Exhibit 1008 - Page 47

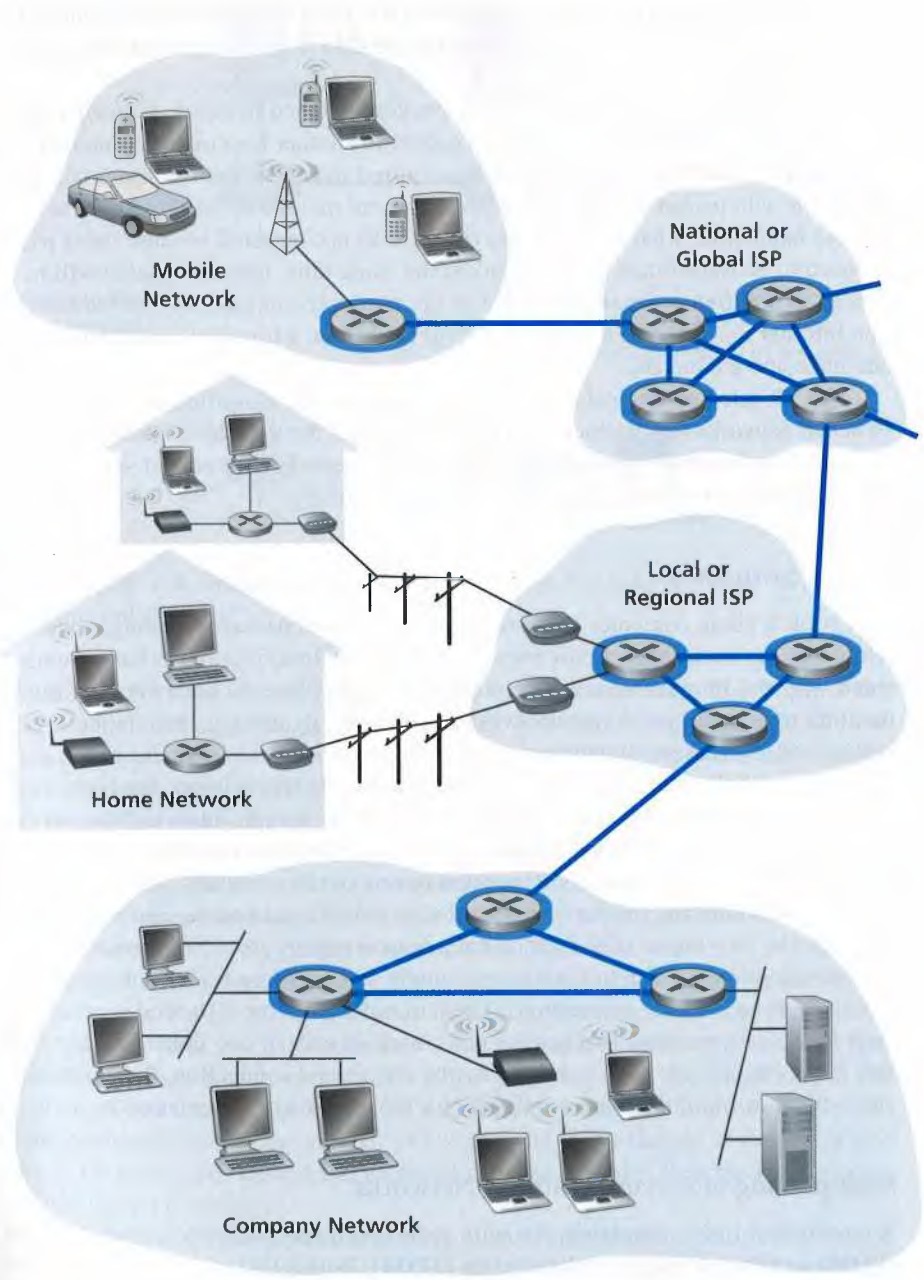


Figure 1.7 ♦ The network core

DELL - Exhibit 1008 - Page 48

duration of the connection. Since bandwidth has been reserved for this sender-to-receiver connection, the sender can transfer the data to the receiver at the *guaranteed* constant rate.

Today's Internet is a quintessential packet-switched network. Consider what happens when one host wants to send a packet to another host over the Internet. As with circuit switching, the packet is transmitted over a series of communication links. But with packet switching, the packet is sent into the network without reserving any bandwidth whatsoever. If one of the links is congested because other packets need to be transmitted over the link at the same time, then our packet will have to wait in a buffer at the sending side of the transmission link, and suffer a delay. The Internet makes its *best effort* to deliver packets in a timely manner, but it does not make any guarantees.

Not all telecommunication networks can be neatly classified as pure circuit-switched networks or pure packet-switched networks. Nevertheless, this fundamental classification into packet- and circuit-switched networks is an excellent starting point in understanding telecommunication network technology.

Circuit Switching

This book is about computer networks, the Internet, and packet switching, not about telephone networks and circuit switching. Nevertheless, it is important to understand why the Internet and other computer networks use packet switching rather than the more traditional circuit-switching technology used in the telephone networks. For this reason, we now give a brief overview of circuit switching.

Figure 1.8 illustrates a circuit-switched network. In this network, the four circuit switches are interconnected by four links. Each of these links has n circuits, so that each link can support n simultaneous connections. The hosts (for example, PCs and workstations) are each directly connected to one of the switches. When two hosts want to communicate, the network establishes a dedicated **end-to-end connection** between the two hosts. (Conference calls between more than two devices are, of course, also possible. But to keep things simple, let's suppose for now that there are only two hosts for each connection.) Thus, in order for Host A to send messages to Host B, the network must first reserve one circuit on each of two links. Because each link has n circuits, for each link used by the end-to-end connection, the connection gets a fraction $1/n$ of the link's bandwidth for the duration of the connection.

Multiplexing in Circuit-Switched Networks

A circuit in a link is implemented with either **frequency-division multiplexing (FDM)** or **time-division multiplexing (TDM)**. With FDM, the frequency spectrum of a link is divided up among the connections established across the link. Specifically, the link dedicates a frequency band to each connection for the duration of the connection. In telephone networks, this frequency band typically has a

DELL - Exhibit 1008 - Page 49

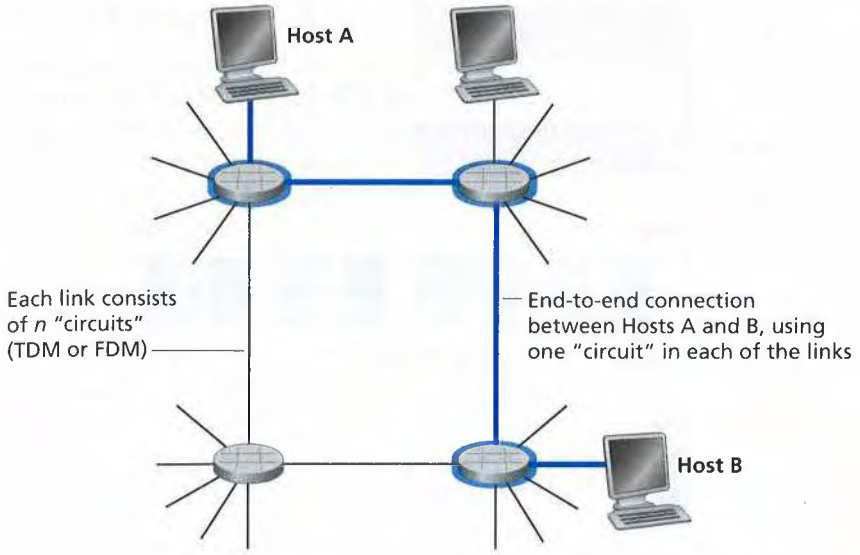


Figure 1.8 ♦ A simple circuit-switched network consisting of four switches and four links

width of 4 kHz (that is, 4,000 hertz or 4,000 cycles per second). The width of the band is called, not surprisingly, the **bandwidth**. FM radio stations also use FDM to share the frequency spectrum between 88 MHz and 108 MHz, with each station being allocated a specific frequency band.

For a TDM link, time is divided into frames of fixed duration, and each frame is divided into a fixed number of time slots. When the network establishes a connection across a link, the network dedicates one time slot in every frame to this connection. These slots are dedicated for the sole use of that connection, with one time slot available for use (in every frame) to transmit the connection's data.

Figure 1.9 illustrates FDM and TDM for a specific network link supporting up to four circuits. For FDM, the frequency domain is segmented into four bands, each of bandwidth 4 kHz. For TDM, the time domain is segmented into frames, with four time slots in each frame; each circuit is assigned the same dedicated slot in the revolving TDM frames. For TDM, the transmission rate of a circuit is equal to the frame rate multiplied by the number of bits in a slot. For example, if the link transmits 8,000 frames per second and each slot consists of 8 bits, then the transmission rate of a circuit is 64 kbps.

Proponents of packet switching have always argued that circuit switching is wasteful because the dedicated circuits are idle during **silent periods**. For example, when one person in a telephone call stops talking, the idle network resources (frequency bands or time slots in the links along the connection's route) cannot be used by other ongoing connections. As another example of how these resources can be

DELL - Exhibit 1008 - Page 50

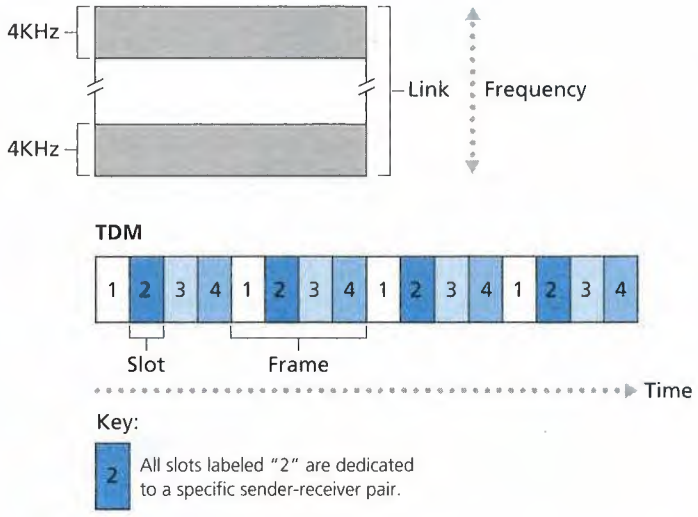


Figure 1.9 ♦ With FDM, each circuit continuously gets a fraction of the bandwidth. With TDM, each circuit gets all of the bandwidth periodically during brief intervals of time (that is, during slots).

underutilized, consider a radiologist who uses a circuit-switched network to remotely access a series of x-rays. The radiologist sets up a connection, requests an image, contemplates the image, and then requests a new image. Network resources are allocated to the connection but are not used (i.e., are wasted) during the radiologist's contemplation periods. Proponents of packet switching also enjoy pointing out that establishing end-to-end circuits and reserving end-to-end bandwidth is complicated and requires complex signaling software to coordinate the operation of the switches along the end-to-end path.

Before we finish our discussion of circuit switching, let's work through a numerical example that should shed further insight on the topic. Let us consider how long it takes to send a file of 640,000 bits from Host A to Host B over a circuit-switched network. Suppose that all links in the network use TDM with 24 slots and have a bit rate of 1.536 Mbps. Also suppose that it takes 500 msec to establish an end-to-end circuit before Host A can begin to transmit the file. How long does it take to send the file? Each circuit has a transmission rate of (1.536 Mbps)/24 = 64 kbps, so it takes (640,000 bits)/(64 kbps) = 10 seconds to transmit the file. To this 10 seconds we add the circuit establishment time, giving 10.5 seconds to send the file. Note that the transmission time is independent of the number of links: The transmission time would be 10 seconds if the end-to-end circuit passed through one link or a hundred links. (The actual end-to-end delay also includes a propagation delay; see Section 1.4.)

DELL - Exhibit 1008 - Page 51

Packet Switching

Distributed applications exchange **messages** in accomplishing their task. Messages can contain anything the protocol designer wants. Messages may perform a control function (for example, the "Hi" messages in our handshaking example) or can contain data, such as an e-mail message, a JPEG image, or an MP3 audio file. In modern computer networks, the source breaks long messages into smaller chunks of data known as **packets**. Between source and destination, each of these packets travels through communication links and **packet switches** (for which there are two predominant types, routers and link-layer switches). Packets are transmitted over each communication link at a rate equal to the *full* transmission rate of the link.

Most packet switches use **store-and-forward transmission** at the inputs to the links. Store-and-forward transmission means that the switch must receive the entire packet before it can begin to transmit the first bit of the packet onto the outbound link. Thus store-and-forward packet switches introduce a store-and-forward delay at the input to each link along the packet's route. Consider how long it takes to send a packet of L bits from one host to another host across a packet-switched network. Let's suppose that there are Q links between the two hosts, each of rate R bps. Assume that this is the only packet in the network. The packet must first be transmitted onto the first link emanating from Host A; this takes L/R seconds. It must then be transmitted on each of the $Q - 1$ remaining links; that is, it must be stored and forwarded $Q - 1$ times, each time with a store-and-forward delay of L/R. Thus the total delay is QL/R.

Each packet switch has multiple links attached to it. For each attached link, the packet switch has an **output buffer** (also called an **output queue**), which stores packets that the router is about to send into that link. The output buffers play a key role in packet switching. If an arriving packet needs to be transmitted across a link but finds the link busy with the transmission of another packet, the arriving packet must wait in the output buffer. Thus, in addition to the store-and-forward delays, packets suffer output buffer **queuing delays**. These delays are variable and depend on the level of congestion in the network. Since the amount of buffer space is finite, an arriving packet may find that the buffer is completely filled with other packets waiting for transmission. In this case, **packet loss** will occur—either the arriving packet or one of the already-queued packets will be dropped. Returning to our restaurant analogy from earlier in this section, the queuing delay is analogous to the amount of time you spend waiting at the restaurant's bar for a table to become free. Packet loss is analogous to being told by the waiter that you must leave the premises because there are already too many other people waiting at the bar for a table.

Figure 1.10 illustrates a simple packet-switched network. In this and subsequent figures, packets are represented by three-dimensional slabs. The width of a slab represents the number of bits in the packet. In this figure, all packets have the same width and hence the same length. Suppose Hosts A and B are sending packets to Host E. Hosts A and B first send their packets along 10 Mbps Ethernet links to the first packet switch. The packet switch then directs these packets to the 1.5 Mbps

DELL - Exhibit 1008 - Page 52

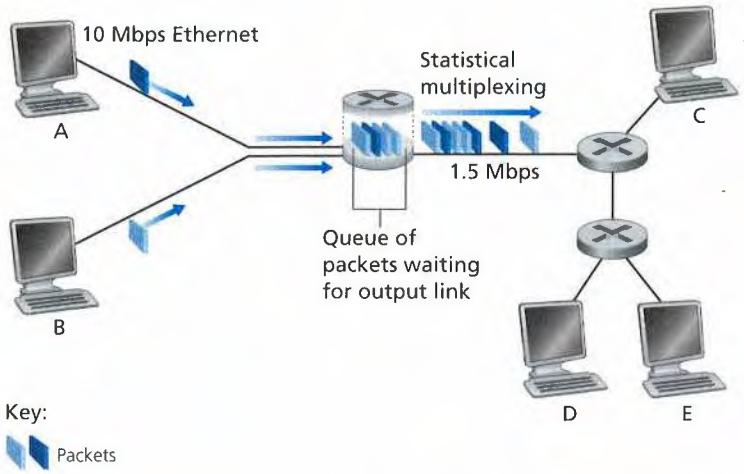


Figure 1.10 ♦ Packet switching

link. If the arrival rate of packets to the switch exceeds the rate at which the switch can forward packets across the 1.5 Mbps output link, congestion will occur as packets queue in the link's output buffer before being transmitted onto the link. We'll examine this queuing delay in more detail in Section 1.4.

Packet Switching Versus Circuit Switching: Statistical Multiplexing

Having described circuit switching and packet switching, let us compare the two. Critics of packet switching have often argued that packet switching is not suitable for real-time services (for example, telephone calls and video conference calls) because of its variable and unpredictable end-to-end delays (due primarily to variable and unpredictable queuing delays). Proponents of packet switching argue that (1) it offers better sharing of bandwidth than circuit switching and (2) it is simpler, more efficient, and less costly to implement than circuit switching. An interesting discussion of packet switching versus circuit switching is [Molinero-Fernandez 2002]. Generally speaking, people who do not like to hassle with restaurant reservations prefer packet switching to circuit switching.

Why is packet switching more efficient? Let's look at a simple example. Suppose users share a 1 Mbps link. Also suppose that each user alternates between periods of activity, when a user generates data at a constant rate of 100 kbps, and periods of inactivity, when a user generates no data. Suppose further that a user is active only 10 percent of the time (and is idly drinking coffee during the remaining 90 percent of the time). With circuit switching, 100 kbps must be *reserved* for *each* user at all times. For example, with circuit-switched TDM, if a one-second frame is divided

DELL - Exhibit 1008 - Page 53

into 10 time slots of 100 ms each, then each user would be allocated one time slot per frame.

Thus, the circuit-switched link can support only 10 (= 1 Mbps/100 kbps) simultaneous users. With packet switching, the probability that a specific user is active is 0.1 (that is, 10 percent). If there are 35 users, the probability that there are 11 or more simultaneously active users is approximately 0.0004. (Homework Problem P7 outlines how this probability is obtained.) When there are 10 or fewer simultaneously active users (which happens with probability 0.9996), the aggregate arrival rate of data is less than or equal to 1 Mbps, the output rate of the link. Thus, when there are 10 or fewer active users, users' packets flow through the link essentially without delay, as is the case with circuit switching. When there are more than 10 simultaneously active users, then the aggregate arrival rate of packets exceeds the output capacity of the link, and the output queue will begin to grow. (It continues to grow until the aggregate input rate falls back below 1 Mbps, at which point the queue will begin to diminish in length.) Because the probability of having more than 10 simultaneously active users is minuscule in this example, packet switching provides essentially the same performance as circuit switching, *but does so while allowing for more than three times the number of users.*

Let's now consider a second simple example. Suppose there are 10 users and that one user suddenly generates one thousand 1,000-bit packets, while other users remain quiescent and do not generate packets. Under TDM circuit switching with 10 slots per frame and each slot consisting of 1,000 bits, the active user can only use its one time slot per frame to transmit data, while the remaining nine times slots in each frame remain idle. It will be 10 seconds before all of the active user's one million bits of data has been transmitted. In the case of packet switching, the active user can continuously send its packets at the full link rate of 1 Mbps, since there are no other users generating packets that need to be multiplexed with the active user's packets. In this case, all of the active user's data will be transmitted within 1 second.

The above examples illustrate two ways in which the performance of packet switching can be superior to that of circuit switching. They also highlight the crucial difference between the two forms of sharing a link's transmission rate among multiple data streams. Circuit switching pre-allocates use of the transmission link regardless of demand, with allocated but unneeded link time going unused. Packet switching on the other hand allocates link use *on demand.* Link transmission capacity will be shared on a packet-by-packet basis only among those users who have packets that need to be transmitted over the link. Such on-demand (rather than pre-allocated) sharing of resources is sometimes referred to as the **statistical multiplexing** of resources.

Although packet switching and circuit switching are both prevalent in today's telecommunication networks, the trend has certainly been in the direction of packet switching. Even many of today's circuit-switched telephone networks are slowly migrating toward packet switching. In particular, telephone networks often use packet switching for the expensive overseas portion of a telephone call.

DELL - Exhibit 1008 - Page 54

1.3.2 How Do Packets Make Their Way Through Packet-Switched Networks?

Earlier we said that a router takes a packet arriving on one of its attached communication links and forwards that packet on to another of its attached communication links. But how does the router determine the link onto which it should forward the packet? This is actually done in different ways by different types of computer networks. In this introductory chapter, we will describe one popular approach, namely, the approach employed by the Internet.

In the Internet, each packet traversing the network contains the address of the packet's destination in its header. As with postal addresses, this address has a hierarchical structure. When a packet arrives at a router in the network, the router examines a portion of the packet's destination address and forwards the packet to an adjacent router. More specifically, each router has a **forwarding table** that maps destination addresses (or portions of the destination addresses) to outbound links. When a packet arrives at a router, the router examines the address and searches its table using this destination address to find the appropriate outbound link. The router then directs the packet to this outbound link.

We just learned that a router uses a packet's destination address to index a forwarding table and determine the appropriate outbound link. But this statement begs yet another question: how do forwarding tables get set? Are they configured by hand in each and every router, or does the Internet use a more automated procedure? This issue will be studied in depth in Chapter 4. But to whet your appetite here, we'll note now that the Internet has a number of special routing protocols that are used to automatically set the forwarding tables. A routing protocol may, for example, determine the shortest path from each router to each destination and use the shortest path results to configure the forwarding tables in the routers.

The end-to-end routing process is analogous to a car driver who does not use maps but instead prefers to ask for directions. For example, suppose Joe is driving from Philadelphia to 156 Lakeside Drive in Orlando, Florida. Joe first drives to his neighborhood gas station and asks how to get to 156 Lakeside Drive in Orlando, Florida. The gas station attendant extracts the Florida portion of the address and tells Joe that he needs to get onto the interstate highway I-95 South, which has an entrance just next to the gas station. He also tells Joe that once he enters Florida he should ask someone else there. Joe then takes I-95 South until he gets to Jacksonville, Florida, at which point he asks another gas station attendant for directions. The attendant extracts the Orlando portion of the address and tells Joe that he should continue on I-95 to Daytona Beach and then ask someone else. In Daytona Beach another gas station attendant also extracts the Orlando portion of the address and tells Joe that he should take I-4 directly to Orlando. Joe takes I-4 and gets off at the Orlando exit. Joe goes to another gas station attendant, and this time the attendant extracts the Lakeside Drive portion of the address and tells Joe the road he must follow to get to Lakeside Drive. Once Joe reaches Lakeside Drive, he asks a kid on a

DELL - Exhibit 1008 - Page 55

bicycle how to get to his destination. The kid extracts the 156 portion of the address and points to the house. Joe finally reaches his ultimate destination.

In the above analogy, the gas-station attendants and kids on bicycles are analogous to routers. Their forwarding tables, which are in their brains, have been configured by years of experience.

How would you actually like to see the end-to-end route that packets take in the Internet? We now invite you to get your hands dirty by interacting with the Traceroute program, by visiting the site http://www.traceroute.org. (For a discussion of Traceroute, see Section 1.4.)

1.3.3 ISPs and Internet Backbones

We saw earlier that end systems (user PCs, PDAs, Web servers, mail servers, and so on) connect into the Internet via an access network. Recall that the access network may be a wired or wireless local area network (for example, in a company, school, or library), a residential cable modem or DSL network, or a residential ISP (for example, AOL or MSN) that is reached via dial-up modem. But connecting end users and content providers into access networks is only a small piece of solving the puzzle of connecting the hundreds of millions of end-systems and hundreds of thousands of networks that make up the Internet. The Internet is a *network of networks*—understanding this phrase is the key to solving this puzzle.

In the public Internet, access networks situated at the edge of the Internet are connected to the rest of the Internet through a tiered hierarchy of ISPs, as shown in Figure 1.11. Access ISPs (for example, residential cable and DSL networks, dial-up access networks such as AOL, wireless access networks, and company and university ISPs using LANs) are at the bottom of this hierarchy. At the very top of the hierarchy is a relatively small number of so-called **tier-1 ISPs**. In many ways, a tier-1 ISP is the same as any network—it has links and routers and is connected to other networks. In other ways, however, tier-1 ISPs are special. Their link speeds are often 622 Mbps or higher, with the larger tier-1 ISPs having links in the 2.5 to 10 Gbps range; their routers must consequently be able to forward packets at extremely high rates. Tier-1 ISPs are also characterized by being:

- Directly connected to *each* of the other tier-1 ISPs
- Connected to a large number of tier-2 ISPs and other customer networks
- International in coverage

Tier-1 ISPs are also known as **Internet backbone** networks. These include Sprint, Verizon, (previously UUNet/WorldCom), AT&T, NTT, Level3, Qwest, and Cable & Wireless. Interestingly, no group officially sanctions tier-1 status; as the saying goes—if you have to ask if you are a member of a group, you're probably not.

A tier-2 ISP typically has regional or national coverage, and (importantly) connects to only a few of the tier-1 ISPs (see Figure 1.11). Thus, in order to reach a

DELL - Exhibit 1008 - Page 56

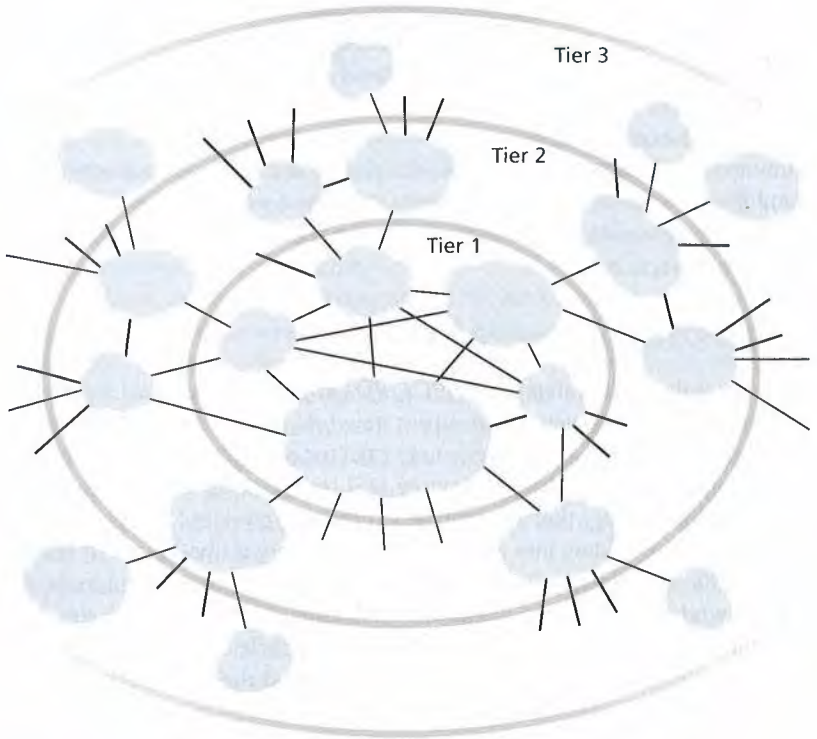


Figure 1.11 ♦ Interconnection of ISPs

large portion of the global Internet, a tier-2 ISP needs to route traffic through one of the tier-1 ISPs to which it is connected. A tier-2 ISP is said to be a **customer** of the tier-1 ISP to which it is connected, and the tier-1 ISP is said to be a **provider** to its customer. Many large companies and institutions connect their enterprise's network directly into a tier-1 or tier-2 ISP, thus becoming a customer of that ISP. A provider ISP charges its customer ISP a fee, which typically depends on the transmission rate of the link connecting the two. A tier-2 network may also choose to connect directly to other tier-2 networks, in which case traffic can flow between the two tier-2 networks without having to pass through a tier-1 network. Below the tier-2 ISPs are the lower-tier ISPs, which connect to the larger Internet via one or more tier-2 ISPs. At the bottom of the hierarchy are the access ISPs. Further complicating matters, some tier-1 providers are also tier-2 providers (that is, vertically integrated), selling Internet access directly to end users and content providers, as well as to lower-tier ISPs. When two ISPs are directly connected to each other, they are said to **peer** with each other. An interesting study [Subramanian 2002] seeks to define the Internet's tiered structure more precisely by studying the Internet's topology in terms of customer-provider and peer-peer relationships.

DELL - Exhibit 1008 - Page 57

Within an ISP's network, the points at which the ISP connects to other ISPs (whether below, above, or at the same level in the hierarchy) are known as **Points of Presence (POPs)**. A POP is simply a group of one or more routers in the ISP's network at which routers in other ISPs or in the networks belonging to the ISP's customers can connect. A tier-1 provider typically has many POPs scattered across different geographical locations in its network, with multiple customer networks and other ISPs connecting into each POP. For a customer network to connect to a provider's POP, the customer typically leases a high-speed link from a third-party telecommunications provider and directly connects one of its routers to a router at the provider's POP. Furthermore, two ISPs may have multiple peering points, connecting with each other at multiple pairs of POPs.

In summary, the topology of the Internet is complex, consisting of dozens of tier-1 and tier-2 ISPs and thousands of lower-tier ISPs. The ISPs are diverse in their coverage, with some spanning multiple continents and oceans, and others limited to narrow regions of the world. The lower-tier ISPs connect to the higher-tier ISPs, and the higher-tier ISPs interconnect with one another. Users and content providers are customers of lower-tier ISPs, and lower-tier ISPs are customers of higher-tier ISPs.

1.4 Delay, Loss, and Throughput in Packet-Switched Networks

Back in Section 1.1 we said that the Internet can be viewed as an infrastructure that provides services to distributed applications running on end systems. Ideally, we would like Internet services to be able to move as much data as we want between any two end systems, instantaneously, without any data loss. Alas, this is a lofty goal, one that is unachievable in reality. Instead, computer networks necessarily constrain throughput (the amount of data per second that can be transferred) between end systems, introduce delays between end systems, and can actually lose packets. On one hand, it is unfortunate that the physical laws of reality introduce delay and loss as well as constrain throughput. On the other hand, because computer networks have these problems, there are many fascinating issues surrounding how to deal with the problems—more than enough issues to fill a course on computer networking and to motivate hundreds of PhD theses! In this section, we'll begin to examine and quantify delay, loss, and throughput in computer networks.

1.4.1 Overview of Delay in Packet-Switched Networks

Recall that a packet starts in a host (the source), passes through a series of routers, and ends its journey in another host (the destination). As a packet travels from one node (host or router) to the subsequent node (host or router) along this path, the

DELL - Exhibit 1008 - Page 58

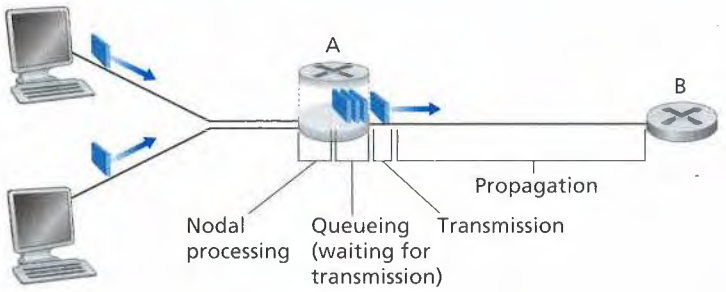


Figure 1.12 ♦ The nodal delay at router A

packet suffers from several types of delays at *each* node along the path. The most important of these delays are the **nodal processing delay, queuing delay, transmission delay,** and **propagation delay**; together, these delays accumulate to give a **total nodal delay**. In order to acquire a deep understanding of packet switching and computer networks, we must understand the nature and importance of these delays.

Types of Delay

Let's explore these delays in the context of Figure 1.12. As part of its end-to-end route between source and destination, a packet is sent from the upstream node through router A to router B. Our goal is to characterize the nodal delay at router A. Note that router A has an outbound link leading to router B. This link is preceded by a queue (also known as a buffer). When the packet arrives at router A from the upstream node, router A examines the packet's header to determine the appropriate outbound link for the packet and then directs the packet to this link. In this example, the outbound link for the packet is the one that leads to router B. A packet can be transmitted on a link only if there is no other packet currently being transmitted on the link and if there are no other packets preceding it in the queue; if the link is currently busy or if there are other packets already queued for the link, the newly arriving packet will then join the queue.

Processing Delay

The time required to examine the packet's header and determine where to direct the packet is part of the **processing delay**. The processing delay can also include other factors, such as the time needed to check for bit-level errors in the packet that occurred in transmitting the packet's bits from the upstream node to router A. Processing delays in high-speed routers are typically on the order of microseconds or less. After this

DELL - Exhibit 1008 - Page 59

nodal processing, the router directs the packet to the queue that precedes the link to router B. (In Chapter 4 we'll study the details of how a router operates.)

Queuing Delay

At the queue, the packet experiences a **queuing delay** as it waits to be transmitted onto the link. The length of the queuing delay of a specific packet will depend on the number of earlier-arriving packets that are queued and waiting for transmission across the link. If the queue is empty and no other packet is currently being transmitted, then our packet's queuing delay will be zero. On the other hand, if the traffic is heavy and many other packets are also waiting to be transmitted, the queuing delay will be long. We will see shortly that the number of packets that an arriving packet might expect to find is a function of the intensity and nature of the traffic arriving at the queue. Queuing delays can be on the order of microseconds to milliseconds in practice.

Transmission Delay

Assuming that packets are transmitted in a first-come-first-served manner, as is common in packet-switched networks, our packet can be transmitted only after all the packets that have arrived before it have been transmitted. Denote the length of the packet by L bits, and denote the transmission rate of the link from router A to router B by R bits/sec. For example, for a 10 Mbps Ethernet link, the rate is $R = 10$ Mbps; for a 100 Mbps Ethernet link, the rate is $R = 100$ Mbps. The **transmission delay** (also called the store-and-forward delay, as discussed in Section 1.3) is L/R. This is the amount of time required to push (that is, transmit) all of the packet's bits into the link. Transmission delays are typically on the order of microseconds to milliseconds in practice.

Propagation Delay

Once a bit is pushed into the link, it needs to propagate to router B. The time required to propagate from the beginning of the link to router B is the **propagation delay**. The bit propagates at the propagation speed of the link. The propagation speed depends on the physical medium of the link (that is, fiber optics, twisted-pair copper wire, and so on) and is in the range of

$$2 \cdot 10^8 \text{ meters/sec to } 3 \cdot 10^8 \text{ meters/sec}$$

which is equal to, or a little less than, the speed of light. The propagation delay is the distance between two routers divided by the propagation speed. That is, the propagation delay is d/s, where d is the distance between router A and router B and s is the propagation speed of the link. Once the last bit of the packet propagates to node B, it and all the preceding bits of the packet are stored in router B. The whole

DELL - Exhibit 1008 - Page 60

process then continues with router B now performing the forwarding. In wide-area networks, propagation delays are on the order of milliseconds.

Comparing Transmission and Propagation Delay

Newcomers to the field of computer networking sometimes have difficulty understanding the difference between transmission delay and propagation delay. The difference is subtle but important. The transmission delay is the amount of time required for the router to push out the packet; it is a function of the packet's length and the transmission rate of the link, but has nothing to do with the distance between the two routers. The propagation delay, on the other hand, is the time it takes a bit to propagate from one router to the next; it is a function of the distance between the two routers, but has nothing to do with the packet's length or the transmission rate of the link.

An analogy might clarify the notions of transmission and propagation delay. Consider a highway that has a tollbooth every 100 kilometers, as shown in Figure 1.13. You can think of the highway segments between tollbooths as links and the tollbooths as routers. Suppose that cars travel (that is, propagate) on the highway at a rate of 100 km/hour (that is, when a car leaves a tollbooth, it instantaneously accelerates to 100 km/hour and maintains that speed between tollbooths). Suppose next that 10 cars, traveling together as a caravan, follow each other in a fixed order. You can think of each car as a bit and the caravan as a packet. Also suppose that each tollbooth services (that is, transmits) a car at a rate of one car per 12 seconds, and that it is late at night so that the caravan's cars are the only cars on the highway. Finally, suppose that whenever the first car of the caravan arrives at a tollbooth, it waits at the entrance until the other nine cars have arrived and lined up behind it. (Thus the entire caravan must be stored at the tollbooth before it can begin to be forwarded.) The time required for the tollbooth to push the entire caravan onto the highway is (10 cars)/(5 cars/minute) = 2 minutes. This time is analogous to the transmission delay in a router. The time required for a car to travel from the exit of one tollbooth to the next tollbooth is 100 km/(100 km/hour) = 1 hour. This time is analogous to propagation delay. Therefore, the time from when the caravan is stored in front of a tollbooth until the caravan is stored in front of the next tollbooth is the sum of transmission delay and propagation delay—in this example, 62 minutes.

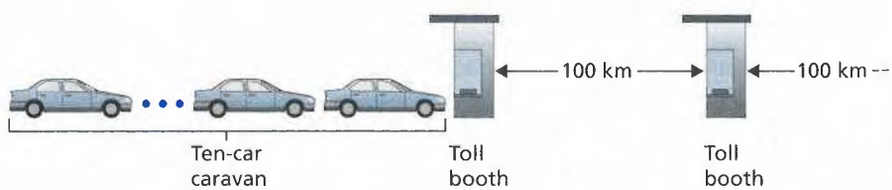


Figure 1.13 ♦ Caravan analogy

DELL - Exhibit 1008 - Page 61

Let's explore this analogy a bit more. What would happen if the tollbooth service time for a caravan were greater than the time for a car to travel between tollbooths? For example, suppose now that the cars travel at the rate of 1,000 km/hour and the tollbooth services cars at the rate of one car per minute. Then the traveling delay between two tollbooths is 6 minutes and the time to serve a caravan is 10 minutes. In this case, the first few cars in the caravan will arrive at the second tollbooth before the last cars in the caravan leave the first tollbooth. This situation also arises in packet-switched networks—the first bits in a packet can arrive at a router while many of the remaining bits in the packet are still waiting to be transmitted by the preceding router.

If a picture speaks a thousand words, then an animation must speak a million words. The companion Web site for this textbook provides an interactive Java applet that nicely illustrates and contrasts transmission delay and propagation delay. The reader is highly encouraged to visit that applet.

If we let d_{proc}, d_{queue}, d_{trans}, and d_{prop} denote the processing, queuing, transmission, and propagation delays, then the total nodal delay is given by

$$d_{nodal} = d_{proc} + d_{queue} + d_{trans} + d_{prop}$$

The contribution of these delay components can vary significantly. For example, d_{prop} can be negligible (for example, a couple of microseconds) for a link connecting two routers on the same university campus; however, d_{prop} is hundreds of milliseconds for two routers interconnected by a geostationary satellite link, and can be the dominant term in d_{nodal}. Similarly, d_{trans} can range from negligible to significant. Its contribution is typically negligible for transmission rates of 10 Mbps and higher (for example, for LANs); however, it can be hundreds of milliseconds for large Internet packets sent over low-speed dial-up modem links. The processing delay, d_{proc}, is often negligible; however, it strongly influences a router's maximum throughput, which is the maximum rate at which a router can forward packets.

1.4.2 Queuing Delay and Packet Loss

The most complicated and interesting component of nodal delay is the queuing delay, d_{queue}. In fact, queuing delay is so important and interesting in computer networking that thousands of papers and numerous books have been written about it [Bertsekas 1991; Daigle 1991; Kleinrock 1975, 1976; Ross 1995]. We give only a high-level, intuitive discussion of queuing delay here; the more curious reader may want to browse through some of the books (or even eventually write a PhD thesis on the subject!). Unlike the other three delays (namely, d_{proc}, d_{trans}, and d_{prop}), the queuing delay can vary from packet to packet. For example, if 10 packets arrive at an empty queue at the same time, the first packet transmitted will suffer no queuing delay, while the last packet transmitted will suffer a relatively large queuing delay (while it waits for the other nine packets to be transmitted). Therefore, when

DELL - Exhibit 1008 - Page 62

characterizing queuing delay, one typically uses statistical measures, such as average queuing delay, variance of queuing delay, and the probability that the queuing delay exceeds some specified value.

When is the queuing delay large and when is it insignificant? The answer to this question depends on the rate at which traffic arrives at the queue, the transmission rate of the link, and the nature of the arriving traffic, that is, whether the traffic arrives periodically or arrives in bursts. To gain some insight here, let a denote the average rate at which packets arrive at the queue (a is in units of packets/sec). Recall that R is the transmission rate; that is, it is the rate (in bits/sec) at which bits are pushed out of the queue. Also suppose, for simplicity, that all packets consist of L bits. Then the average rate at which bits arrive at the queue is La bits/sec. Finally, assume that the queue is very big, so that it can hold essentially an infinite number of bits. The ratio La/R, called the **traffic intensity**, often plays an important role in estimating the extent of the queuing delay. If $La/R > 1$, then the average rate at which bits arrive at the queue exceeds the rate at which the bits can be transmitted from the queue. In this unfortunate situation, the queue will tend to increase without bound and the queuing delay will approach infinity! Therefore, one of the golden rules in traffic engineering is: *Design your system so that the traffic intensity is no greater than 1.*

Now consider the case $La/R \leq 1$. Here, the nature of the arriving traffic impacts the queuing delay. For example, if packets arrive periodically—that is, one packet arrives every L/R seconds—then every packet will arrive at an empty queue and there will be no queuing delay. On the other hand, if packets arrive in bursts but periodically, there can be a significant average queuing delay. For example, suppose N packets arrive simultaneously every $(L/R)N$ seconds. Then the first packet transmitted has no queuing delay; the second packet transmitted has a queuing delay of L/R seconds; and more generally, the nth packet transmitted has a queuing delay of $(n - 1)L/R$ seconds. We leave it as an exercise for you to calculate the average queuing delay in this example.

The two examples of periodic arrivals described above are a bit academic. Typically, the arrival process to a queue is *random;* that is, the arrivals do not follow any pattern and the packets are spaced apart by random amounts of time. In this more realistic case, the quantity La/R is not usually sufficient to fully characterize the queueing delay statistics. Nonetheless, it is useful in gaining an intuitive understanding of the extent of the queuing delay. In particular, if the traffic intensity is close to zero, then packet arrivals are few and far between and it is unlikely that an arriving packet will find another packet in the queue. Hence, the average queuing delay will be close to zero. On the other hand, when the traffic intensity is close to 1, there will be intervals of time when the arrival rate exceeds the transmission capacity (due to variations in packet arrival rate), and a queue will form during these periods of time; when the arrival rate is less than the transmission capacity, the length of the queue will shrink. Nonetheless, as the traffic intensity approaches 1, the average queue length gets larger and larger. The qualitative dependence of average queuing delay on the traffic intensity is shown in Figure 1.14.

DELL - Exhibit 1008 - Page 63

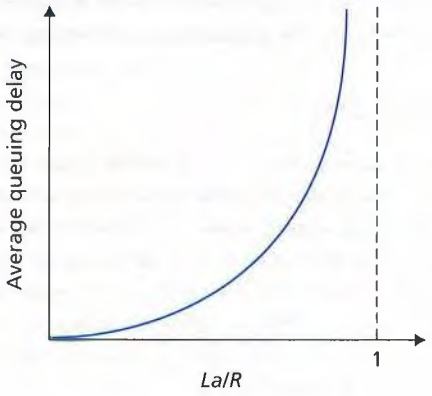


Figure 1.14 ♦ Dependence of average queuing delay on traffic intensity

One important aspect of Figure 1.14 is the fact that as the traffic intensity approaches 1, the average queuing delay increases rapidly. A small percentage increase in the intensity will result in a much larger percentage-wise increase in delay. Perhaps you have experienced this phenomenon on the highway. If you regularly drive on a road that is typically congested, the fact that the road is typically congested means that its traffic intensity is close to 1. If some event causes an even slightly larger-than-usual amount of traffic, the delays you experience can be huge.

To really get a good feel for what queuing delays are about, you are encouraged once again to visit the companion Web site, which provides an interactive Java applet for a queue. If you set the packet arrival rate high enough so that the traffic intensity exceeds 1, you will see the queue slowly build up over time.

Packet Loss

In our discussions above, we have assumed that the queue is capable of holding an infinite number of packets. In reality a queue preceding a link has finite capacity, although the queuing capacity greatly depends on the switch design and cost. Because the queue capacity is finite, packet delays do not really approach infinity as the traffic intensity approaches 1. Instead, a packet can arrive to find a full queue. With no place to store such a packet, a router will **drop** that packet; that is, the packet will be **lost**. This overflow at a queue can again be seen in the Java applet for a queue when the traffic intensity is greater than 1.

From an end-system viewpoint, a packet loss will look like a packet having been transmitted into the network core but never emerging from the network at the destination. The fraction of lost packets increases as the traffic intensity increases. Therefore, performance at a node is often measured not only in terms of delay, but also in terms of the probability of packet loss. As we'll discuss in the subsequent

DELL - Exhibit 1008 - Page 64

chapters, a lost packet may be retransmitted on an end-to-end basis in order to ensure that all data are eventually transferred from source to destination

1.4.3 End-to-End Delay

Our discussion up to this point has focused on the nodal delay, that is, the delay at a single router. Let's now consider the total delay from source to destination. To get a handle on this concept, suppose there are $N - 1$ routers between the source host and the destination host. Let's also suppose for the moment that the network is uncongested (so that queuing delays are negligible), the processing delay at each router and at the source host is d_{proc}, the transmission rate out of each router and out of the source host is R bits/sec, and the propagation on each link is d_{prop}. The nodal delays accumulate and give an end-to-end delay,

$$d_{end\text{-}end} = N\,(d_{proc} + d_{trans} + d_{prop})$$

where, once again, $d_{trans} = L/R$, where L is the packet size. We leave it to you to generalize this formula to the case of heterogeneous delays at the nodes and to the presence of an average queuing delay at each node.

Traceroute

To get a hands-on feel for end-to-end delay in a computer network, we can make use of the Traceroute program. Traceroute is a simple program that can run in any Internet host. When the user specifies a destination hostname, the program in the source host sends multiple, special packets toward that destination. As these packets work their way toward the destination, they pass through a series of routers. When a router receives one of these special packets, it sends back to the source a short message that contains the name and address of the router.

More specifically, suppose there are $N - 1$ routers between the source and the destination. Then the source will send N special packets into the network, with each packet addressed to the ultimate destination. These N special packets are marked *1* through N, with the first packet marked *1* and the last packet marked N. When the nth router receives the nth packet marked n, the router does not forward the packet toward its destination, but instead sends a message back to the source. When the destination host receives the Nth packet, it too returns a message back to the source. The source records the time that elapses between when it sends a packet and when it receives the corresponding return message; it also records the name and address of the router (or the destination host) that returns the message. In this manner, the source can reconstruct the route taken by packets flowing from source to destination, and the source can determine the round-trip delays to all the intervening routers. Traceroute actually repeats the experiment just described three times, so the source actually sends $3 \cdot N$ packets to the destination. RFC 1393 describes Traceroute in detail.

DELL - Exhibit 1008 - Page 65

Here is an example of the output of the Traceroute program, where the route was being traced from the source host gaia.cs.umass.edu (at the University of Massachusetts) to the host cis.poly.edu (at Polytechnic University in Brooklyn). The output has six columns: the first column is the n value described above, that is, the number of the router along the route; the second column is the name of the router; the third column is the address of the router (of the form xxx.xxx.xxx.xxx); the last three columns are the round-trip delays for three experiments. If the source receives fewer than three messages from any given router (due to packet loss in the network), Traceroute places an asterisk just after the router number and reports fewer than three round-trip times for that router.

```
1 cs-gw (128.119.240.254) 1.009 ms 0.899 ms 0.993 ms
2 128.119.3.154 (128.119.3.154) 0.931 ms 0.441 ms 0.651 ms
3 border4-rt-gi-1-3.gw.umass.edu (128.119.2.194) 1.032 ms 0.484 ms 0.451 ms
4 acr1-ge-2-1-0.Boston.cw.net (208.172.51.129) 10.006 ms 8.150 ms 8.460 ms
5 agr4-loopback.NewYork.cw.net (206.24.194.104) 12.272 ms 14.344 ms 13.267 ms
6 acr2-loopback.NewYork.cw.net (206.24.194.62) 13.225 ms 12.292 ms 12.148 ms
7 pos10-2.core2.NewYork1.Level3.net (209.244.160.133) 12.218 ms 11.823 ms 11.793 ms
8 gige9-1-52.hsipaccess1.NewYork1.Level3.net (64.159.17.39) 13.081 ms 11.556 ms 13.297 ms
9 p0-0.polyu.bbnplanet.net (4.25.109.122) 12.716 ms 13.052 ms 12.786 ms
10 cis.poly.edu (128.238.32.126) 14.080 ms 13.035 ms 12.802 ms
```

In the trace above there are nine routers between the source and the destination. Most of these routers have a name, and all of them have addresses. For example, the name of Router 3 is `border4-rt-gi-1-3.gw.umass.edu` and its address is `128.119.2.194`. Looking at the data provided for this same router, we see that in the first of the three trials the round-trip delay between the source and the router was 1.03 msec. The round-trip delays for the subsequent two trials were 0.48 and 0.45 msec. These round-trip delays include all of the delays just discussed, including transmission delays, propagation delays, router processing delays, and queuing delays. Because the queuing delay is varying with time, the round-trip delay of packet n sent to a router n can sometimes be longer than the round-trip delay of packet $n+1$ sent to router $n+1$. Indeed, we observe this phenomena in the above example: the delays to Router 6 are larger than the delays to Router 7!

Want to try out Traceroute for yourself? We *highly* recommended that you visit http://www.traceroute.org, which provides a Web interface to an extensive list of sources for route tracing. You choose a source and supply the hostname for any destination. The Traceroute program then does all the work. There are a number of free software programs that provide a graphical interface to Traceroute; one of our favorite is PingPlotter [PingPlotter 2007].

End System, Application, and Other Delays

In addition to processing, transmission, and propagation delays, there can be additional significant delays in the end systems. For example, dial-up modems introduce

DELL - Exhibit 1008 - Page 66

a modulation/encoding delay, which can be on the order of tens of milliseconds. (The modulation/encoding delays for other access technologies—including Ethernet, cable modem, and DSL—are less significant and usually negligible.) An end system wanting to transmit a packet into a shared medium (e.g., as in a WiFi or Ethernet scenario) may *purposefully* delay its transmission as part of its protocol for sharing the medium with other end systems; we'll consider such protocols in detail in Chapter 5. Another important delay is media packetization delay, which is present in Voice-over-IP (VoIP) applications. In VoIP, the sending side must first fill a packet with encoded digitized speech before passing the packet to the Internet. This time to fill a packet—called the packetization delay—can be significant and can impact the user-perceived quality of a VoIP call. This issue will be further explored in a homework problem at the end of this chapter.

1.4.4 Throughput in Computer Networks

In addition to delay and packet loss, another critical performance measure in computer networks is end-to-end throughput. To define throughput, consider transferring a large file from Host A to Host B across a computer network. This transfer might be, for example, a large video clip from one peer to another in a P2P file sharing system. The **instantaneous throughput** at any instant of time is the rate (in bits/sec) at which Host B is receiving the file. (Many applications, including many P2P file sharing systems, display the instantaneous throughput during downloads in the user interface—perhaps you have observed this before!) If the file consists of F bits and the transfer takes T seconds for Host B to receive all F bits, then the **average throughput** of the file transfer is F/T bits/sec. For some applications, such as Internet telephony, it is desirable to have a low delay and an instantaneous throughput consistently above some threshold (for example, over 24 kbps for some Internet telephony applications and over 256 kbps for some real-time video applications). For other applications, including those involving file transfers, delay is not critical, but it is desirable to have the highest possible throughput.

To gain further insight into the important concept of throughput, let's consider a few examples. Figure 1.15(a) shows two end systems, a server and a client, connected by two communication links and a router. Consider the throughput for a file transfer from the server to the client. Let R_s denote the rate of the link between the server and the router; and R_c denote the rate of the link between the router and the client. Suppose that the only bits being sent in the entire network are those from the server to the client. We now ask, in this ideal scenario, what is the server-to-client throughput? To answer this question, we may think of bits as *fluid* and communication links as *pipes*. Clearly, the server cannot pump bits through its link at a rate faster than R_s bps; and the router cannot forward bits at a rate faster than R_c bps. If $R_s < R_c$, then the bits pumped by the server will "flow" right through the router and arrive at the client at a rate of R_s bps, giving a throughput of R_s bps. If, on the other hand, $R_c < R_s$, then the router will not be able to forward bits as quickly as

DELL - Exhibit 1008 - Page 67

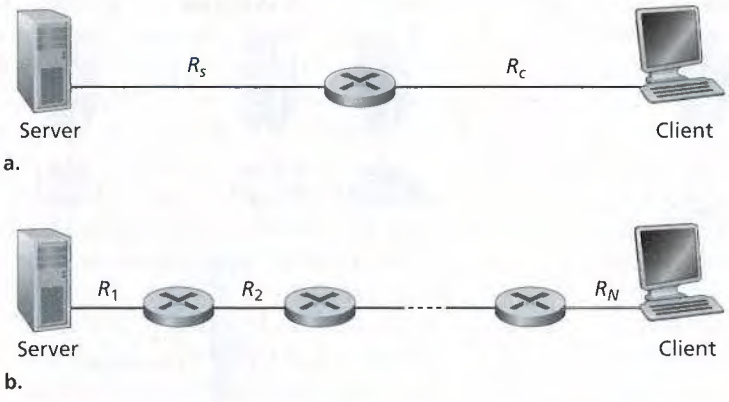


Figure 1.15 ♦ Throughput for a file transfer from server to client

it receives them. In this case, bits will only leave the router at rate R_c, giving an end-to-end throughput of R_c. (Note also that if bits continue to arrive at the router at rate R_s, and continue to leave the router at R_c, the backlog of bits at the router waiting for transmission to the client will grow and grow—a most undesirable situation!) Thus, for this simple two-link network, the throughput is $\min\{R_c, R_s\}$, that is, it is the transmission rate of the **bottleneck link**. Having determined the throughput, we can now approximate the time it takes to transfer a large file of F bits from server to client as $F/\min\{R_s, R_c\}$. For a specific example, suppose you are downloading an MP3 file of $F = 32$ million bits, the server has a transmission rate of $R_s = 2$ Mbps, and you have an access link of $R_c = 1$ Mbps. The time needed to transfer the file is then 32 seconds. Of course, these expressions for throughput and transfer time are only approximations, as they do not account for packet-level and protocol issues.

Figure 1.15(b) now shows a network with N links between the server and the client, with the transmission rates of the N links being $R_1, R_2, \ldots, R_N$. Applying the same analysis as for the two-link network, we find that the throughput for a file transfer from server to client is $\min\{R_1, R_2, \ldots, R_N\}$, which is once again the transmission rate of the bottleneck link along the path between server and client.

Now consider another example motivated by today's Internet. Figure 1.16(a) shows two end systems, a server and a client, connected to a computer network. Consider the throughput for a file transfer from the server to the client. The server is connected to the network with an access link of rate R_s and the client is connected to the network with an access link of rate R_c. Now suppose that all the links in the core of the communication network have very high transmission rates, much higher than R_s and R_c. Indeed, today, the core of the Internet is over-provisioned with high speed links that experience little congestion [Akella 2003]. Also suppose that the only bits being sent in the entire network are those from the server to the client. Because the core of the computer network is like a wide pipe in this example, the rate at which

DELL - Exhibit 1008 - Page 68

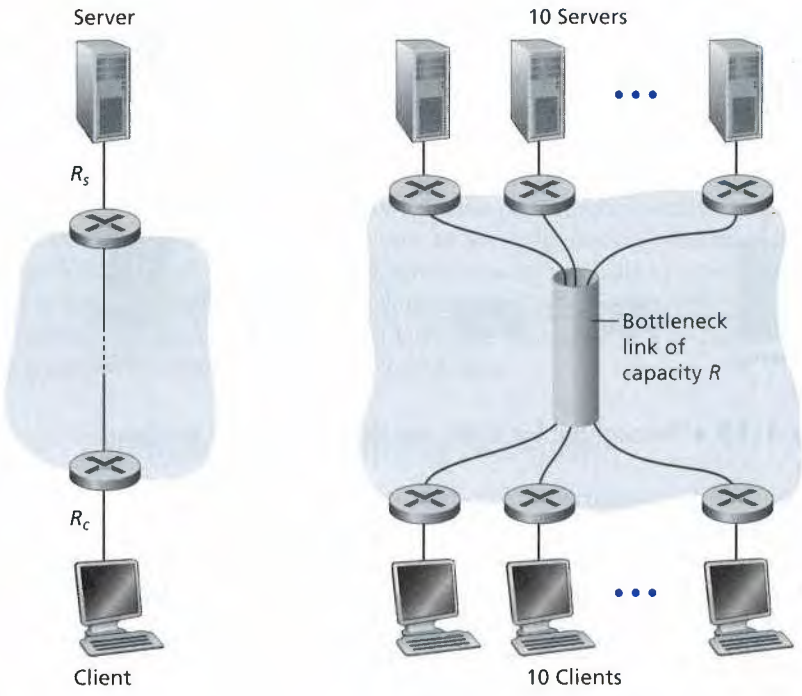


Figure 1.16 ♦ End-to-end throughput: (a) Client downloads a file from server; (b) 10 clients downloading with 10 servers.

bits can flow from source to destination is again the minimum of R_s and R_c, that is, throughput = $\min\{R_s, R_c\}$. Therefore, the constraining factor for throughput in today's Internet is the typically access network.

For a final example, consider Figure 1.16(b) in which there are 10 servers and 10 clients connected to the core of the computer network. In this example, there are 10 simultaneous downloads taking place, involving 10 client-server pairs. Suppose that these 10 downloads are the only traffic in the network at the current time. As shown in the figure, there is a link in the core that is traversed by all 10 downloads. Denote R for the transmission rate of this link R. Let's suppose that all server access links have the same rate R_s, all client access links have the same rate R_c, and the transmission rates of all the links in the core—except the one common link of rate R—are much larger than R_s, R_c, and R. Now we ask, what are the throughputs of the downloads? Clearly, if the rate of the common link, R, is large—say a hundred times larger than both R_s and R_c—then the throughput for each download will once again be $\min\{R_s, R_c\}$. But what if the rate of the common link is of the same order as R_s and R_c? What will the throughput be in this case? Let's take a look at a specific

DELL - Exhibit 1008 - Page 69

example. Suppose R_s = 2 Mbps, R_c = 1 Mbps, R = 5 Mbps, and the common link divides its transmission rate equally among the 10 downloads. Then the bottleneck for each download is no longer in the access network, but is now instead the shared link in the core, which only provides each download with 500 kbps of throughput. Thus the end-to-end throughput for each download is now reduced to 500 kbps.

The examples in Figure 1.15 and Figure 1.16(a) show that throughput depends on the transmission rates of the links over which the data flows. We saw that when there is no other intervening traffic, the throughput can simply be approximated as the minimum transmission rate along the path between source and destination. The example in Figure 1.16(b) shows that more generally the throughput depends not only on the transmission rates of the links along the path, but also on the intervening traffic. In particular, a link with a high transmission rate may nonetheless be the bottleneck link for a file transfer if many other data flows are also passing through that link. We will examine throughput in computer networks more closely in the homework problems and in the subsequent chapters.

1.5 Protocol Layers and Their Service Models

From our discussion thus far, it is apparent that the Internet is an *extremely* complicated system. We have seen that there are many pieces to the Internet: numerous applications and protocols, various types of end systems, packet switches, and various types of link-level media. Given this enormous complexity, is there any hope of organizing a network architecture, or at least our discussion of network architecture? Fortunately, the answer to both questions is yes.

1.5.1 Layered Architecture

Before attempting to organize our thoughts on Internet architecture, let's look for a human analogy. Actually, we deal with complex systems all the time in our everyday life. Imagine if someone asked you to describe, for example, the airline system. How would you find the structure to describe this complex system that has ticketing agents, baggage checkers, gate personnel, pilots, airplanes, air traffic control, and a worldwide system for routing airplanes? One way to describe this system might be to describe the series of actions you take (or others take for you) when you fly on an airline. You purchase your ticket, check your bags, go to the gate, and eventually get loaded onto the plane. The plane takes off and is routed to its destination. After your plane lands, you deplane at the gate and claim your bags. If the trip was bad, you complain about the flight to the ticket agent (getting nothing for your effort). This scenario is shown in Figure 1.17.

Already, we can see some analogies here with computer networking: You are being shipped from source to destination by the airline; a packet is shipped from

DELL - Exhibit 1008 - Page 70

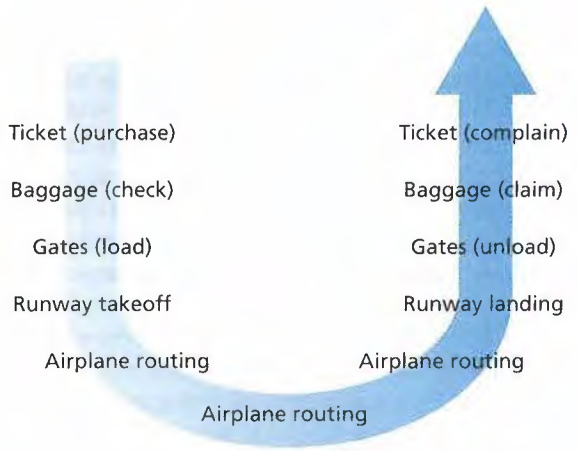


Figure 1.17 ♦ Taking an airplane trip: actions

source host to destination host in the Internet. But this is not quite the analogy we are after. We are looking for some *structure* in Figure 1.17. Looking at Figure 1.17, we note that there is a ticketing function at each end; there is also a baggage function for already-ticketed passengers, and a gate function for already-ticketed and already-baggage-checked passengers. For passengers who have made it through the gate (that is, passengers who are already ticketed, baggage-checked, and through the gate), there is a takeoff and landing function, and while in flight, there is an airplane-routing function. This suggests that we can look at the functionality in Figure 1.17 in a *horizontal* manner, as shown in Figure 1.18.

Figure 1.18 has divided the airline functionality into layers, providing a framework in which we can discuss airline travel. Note that each layer, combined with the

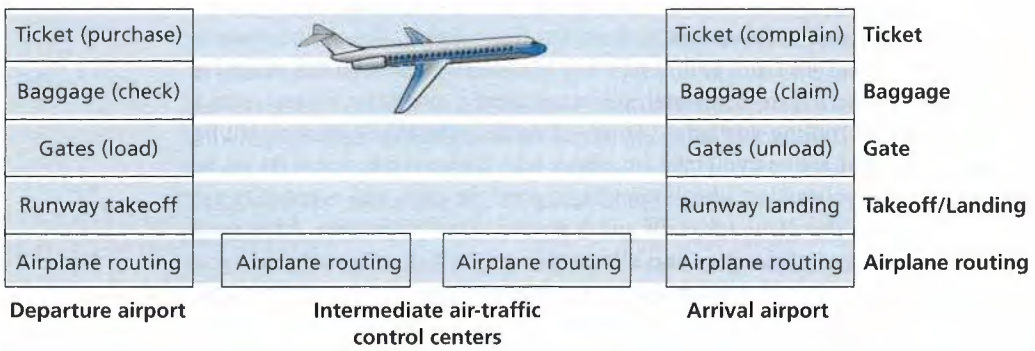


Figure 1.18 ♦ Horizontal layering of airline functionality

DELL - Exhibit 1008 - Page 71

layers below it, implements some functionality, some *service*. At the ticketing layer and below, airline-counter-to-airline-counter transfer of a person is accomplished. At the baggage layer and below, baggage-check-to-baggage-claim transfer of a person and bags is accomplished. Note that the baggage layer provides this service only to an already-ticketed person. At the gate layer, departure-gate-to-arrival-gate transfer of a person and bags is accomplished. At the takeoff/landing layer, runway-to-runway transfer of people and their bags is accomplished. Each layer provides its service by (1) performing certain actions within that layer (for example, at the gate layer, loading and unloading people from an airplane) and by (2) using the services of the layer directly below it (for example, in the gate layer, using the runway-to-runway passenger transfer service of the takeoff/landing layer).

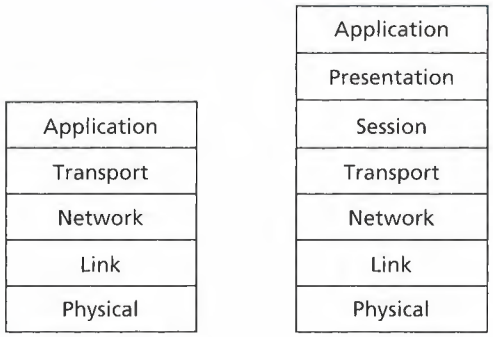
A layered architecture allows us to discuss a well-defined, specific part of a large and complex system. This simplification itself is of considerable value by providing modularity, making it much easier to change the implementation of the service provided by the layer. As long as the layer provides the same service to the layer above it, and uses the same services from the layer below it, the remainder of the system remains unchanged when a layer's implementation is changed. (Note that changing the implementation of a service is very different from changing the service itself!) For example, if the gate functions were changed (for instance, to have people board and disembark by height), the remainder of the airline system would remain unchanged since the gate layer still provides the same function (loading and unloading people); it simply implements that function in a different manner after the change. For large and complex systems that are constantly being updated, the ability to change the implementation of a service without affecting other components of the system is another important advantage of layering.

Protocol Layering

But enough about airlines. Let's now turn our attention to network protocols. To provide structure to the design of network protocols, network designers organize protocols—and the network hardware and software that implement the protocols—in **layers**. Each protocol belongs to one of the layers, just as each function in the airline architecture in Figure 1.18 belonged to a layer. We are again interested in the **services** that a layer offers to the layer above—the so-called **service model** of a layer. Just as in the case of our airline example, each layer provides its service by (1) performing certain actions within that layer and by (2) using the services of the layer directly below it. For example, the services provided by layer n may include reliable delivery of messages from one edge of the network to the other. This might be implemented by using an unreliable edge-to-edge message delivery service of layer $n - 1$, and adding layer n functionality to detect and retransmit lost messages.

A protocol layer can be implemented in software, in hardware, or in a combination of the two. Application-layer protocols—such as HTTP and SMTP—are almost

DELL - Exhibit 1008 - Page 72



a. Five-layer Internet protocol stack

b. Seven-layer ISO OSI reference model

Figure 1.19 ♦ The Internet protocol stack (a) and OSI reference model (b)

always implemented in software in the end systems; so are transport-layer protocols. Because the physical layer and data link layers are responsible for handling communication over a specific link, they are typically implemented in a network interface card (for example, Ethernet or WiFi interface cards) associated with a given link. The network layer is often a mixed implementation of hardware and software. Also note that just as the functions in the layered airline architecture were distributed among the various airports and flight control centers that make up the system, so too is a layer *n* protocol *distributed* among the end systems, packet switches, and other components that make up the network. That is, there's often a piece of a layer *n* protocol in each of these network components.

Protocol layering has conceptual and structural advantages. As we have seen, layering provides a structured way to discuss system components. Modularity makes it easier to update system components. We mention, however, that some researchers and networking engineers are vehemently opposed to layering [Wakeman 1992]. One potential drawback of layering is that one layer may duplicate lower-layer functionality. For example, many protocol stacks provide error recovery on both a per-link basis and an end-to-end basis. A second potential drawback is that functionality at one layer may need information (for example, a timestamp value) that is present only in another layer; this violates the goal of separation of layers.

When taken together, the protocols of the various layers are called the **protocol stack**. The Internet protocol stack consists of five layers: the physical, link, network, transport, and application layers, as shown in Figure 1.19(a). If you examine the Table of Contents, you will see that we have roughly organized this book using the layers of the Internet protocol stack. We take a **top-down approach**, first covering the application layer and then proceeding downward.

DELL - Exhibit 1008 - Page 73

Application Layer

The application layer is where network applications and their application-layer protocols reside. The Internet's application layer includes many protocols, such as the HTTP protocol (which provides for Web document request and transfer), SMTP (which provides for the transfer of e-mail messages), and FTP (which provides for the transfer of files between two end systems). We'll see that certain network functions, such as the translation of human-friendly names for Internet end systems like www.ietf.org to a 32-bit network address, are also done with the help of a specific application-layer protocol, namely, the domain name system (DNS). We'll see in Chapter 2 that it is very easy to create and deploy our own new application-layer protocols.

An application-layer protocol is distributed over multiple end systems, with the application in one end system using the protocol to exchange packets of information with the application in another end system. We'll refer to this packet of information at the application layer as a **message**.

Transport Layer

The Internet's transport layer transports application-layer messages between application endpoints. In the Internet there are two transport protocols, TCP and UDP, either of which can transport application-layer messages. TCP provides a connection-oriented service to its applications. This service includes guaranteed delivery of application-layer messages to the destination and flow control (that is, sender/receiver speed matching). TCP also breaks long messages into shorter segments and provides a congestion-control mechanism, so that a source throttles its transmission rate when the network is congested. The UDP protocol provides a connectionless service to its applications. This is a no-frills service that provides no reliability, no flow control, and no congestion control. In this book, we'll refer to a transport-layer packet as a **segment**.

Network Layer

The Internet's network layer is responsible for moving network-layer packets known as **datagrams** from one host to another. The Internet transport-layer protocol (TCP or UDP) in a source host passes a transport-layer segment and a destination address to the network layer, just as you would give the postal service a letter with a destination address. The network layer then provides the service of delivering the segment to the transport layer in the destination host.

The Internet's network layer includes the celebrated IP Protocol, which defines the fields in the datagram as well as how the end systems and routers act on these fields. There is only one IP protocol, and all Internet components that have a network layer must run the IP protocol. The Internet's network layer also contains routing protocols that determine the routes that datagrams take between sources and

DELL - Exhibit 1008 - Page 74

destinations. The Internet has many routing protocols. As we saw in Section 1.3, the Internet is a network of networks, and within a network, the network administrator can run any routing protocol desired. Although the network layer contains both the IP protocol and numerous routing protocols, it is often simply referred to as the IP layer, reflecting the fact that IP is the glue that binds the Internet together.

Link Layer

The Internet's network layer routes a datagram through a series of routers between the source and destination. To move a packet from one node (host or router) to the next node in the route, the network layer relies on the services of the link layer. In particular, at each node, the network layer passes the datagram down to the link layer, which delivers the datagram to the next node along the route. At this next node, the link layer passes the datagram up to the network layer.

The services provided by the link layer depend on the specific link-layer protocol that is employed over the link. For example, some link-layer protocols provide reliable delivery, from transmitting node, over one link, to receiving node. Note that this reliable delivery service is different from the reliable delivery service of TCP, which provides reliable delivery from one end system to another. Examples of link-layer protocols include Ethernet, WiFi, and the Point-to-Point Protocol (PPP). As datagrams typically need to traverse several links to travel from source to destination, a datagram may be handled by different link-layer protocols at different links along its route. For example, a datagram may be handled by Ethernet on one link and by PPP on the next link. The network layer will receive a different service from each of the different link-layer protocols. In this book, we'll refer to the link-layer packets as **frames**.

Physical Layer

While the job of the link layer is to move entire frames from one network element to an adjacent network element, the job of the physical layer is to move the *individual bits* within the frame from one node to the next. The protocols in this layer are again link dependent and further depend on the actual transmission medium of the link (for example, twisted-pair copper wire, single-mode fiber optics). For example, Ethernet has many physical-layer protocols: one for twisted-pair copper wire, another for coaxial cable, another for fiber, and so on. In each case, a bit is moved across the link in a different way.

The OSI Model

Having discussed the Internet protocol stack in detail, we should mention that it is not the only protocol stack around. In particular, back in the late 1970s, the International Organization for Standardization (ISO) proposed that computer networks be

DELL - Exhibit 1008 - Page 75

organized around seven layers, called the Open Systems Interconnection (OSI) model [ISO 2007]. The OSI model took shape when the protocols that were to become the Internet protocols were in their infancy, and were but one of many different protocol suites under development; in fact, the inventors of the original OSI model probably did not have the Internet in mind when creating it. Nevertheless, beginning in the late 1970s, many training and university courses picked up on the ISO mandate and organized courses around the seven-layer model. Because of its early impact on networking education, the seven-layer model continues to linger on in some networking textbooks and training courses.

The seven layers of the OSI reference model, shown in Figure 1.19(b), are: application layer, presentation layer, session layer, transport layer, network layer, data link layer, and physical layer. The functionality of five of these layers is roughly the same as their similarly named Internet counterparts. Thus, let's consider the two additional layers present in the OSI reference model—the presentation layer and the session layer. The role of the presentation layer is to provide services that allow communicating applications to interpret the meaning of data exchanged. These services include data compression and data encryption (which are self-explanatory) as well as data description (which, as we will see in Chapter 9, frees the applications from having to worry about the internal format in which data are represented/stored—formats that may differ from one computer to another). The session layer provides for delimiting and synchronization of data exchange, including the means to build a checkpointing and recovery scheme.

The fact that the Internet lacks two layers found in the OSI reference model poses a couple of interesting questions: Are the services provided by these layers unimportant? What if an application *needs* one of these services? The Internet's answer to both of these questions is the same—it's up to the application developer. It's up to the application developer to decide if a service is important, and if the service *is* important, it's up to the application developer to build that functionality into the application.

1.5.2 Messages, Segments, Datagrams, and Frames

Figure 1.20 shows the physical path that data takes down a sending end system's protocol stack, up and down the protocol stacks of an intervening link-layer switch and router, and then up the protocol stack at the receiving end system. As we discuss later in this book, routers and link-layer switches are both packet switches. Similar to end systems, routers and link-layer switches organize their networking hardware and software into layers. But routers and link-layer switches do not implement *all* of the layers in the protocol stack; they typically implement only the bottom layers. As shown in Figure 1.20, link-layer switches implement layers 1 and 2; routers implement layers 1 through 3. This means, for example, that Internet routers are capable of implementing the IP protocol (a layer 3 protocol), while link-layer switches are not. We'll see later that while link-layer switches do not recognize IP addresses, they

DELL - Exhibit 1008 - Page 76

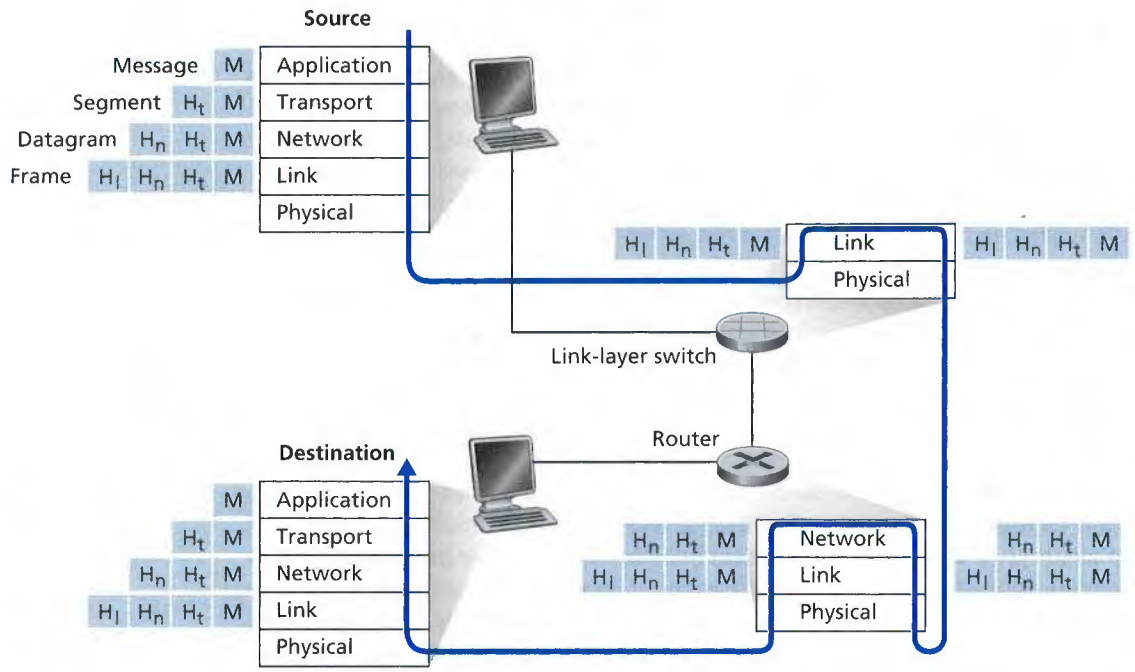


Figure 1.20 ♦ Hosts, routers, and link-layer switches; each contains a different set of layers, reflecting their differences in functionality.

are capable of recognizing layer 2 addresses, such as Ethernet addresses. Note that hosts implement all five layers; this is consistent with the view that the Internet architecture puts much of its complexity at the edges of the network.

Figure 1.20 also illustrates the important concept of **encapsulation**. At the sending host, an **application-layer message** (M in Figure 1.20) is passed to the transport layer. In the simplest case, the transport layer takes the message and appends additional information (so-called transport-layer header information, H_t in Figure 1.20) that will be used by the receiver-side transport layer. The application-layer message and the transport-layer header information together constitute the **transport-layer segment**. The transport-layer segment thus encapsulates the application-layer message. The added information might include information allowing the receiver-side transport layer to deliver the message up to the appropriate application, and error-detection bits that allow the receiver to determine whether bits in the message have been changed in route. The transport layer then passes the segment to the network layer, which adds network-layer header information (H_n in Figure 1.20) such as source and destination end system addresses, creating a **network-layer datagram**. The datagram is then passed to the link layer, which (of course!) will add its own link-layer header information and create a **link-layer frame**. Thus, we see that at

DELL - Exhibit 1008 - Page 77

each layer, a packet has two types of fields: header fields and a **payload field**. The payload is typically a packet from the layer above.

A useful analogy here is the sending of an interoffice memo from one corporate branch office to another via the public postal service. Suppose Alice, who is in one branch office, wants to send a memo to Bob, who is in another branch office. The *memo* is analogous to the *application-layer message*. Alice puts the memo in an interoffice envelope with Bob's name and department written on the front of the envelope. The *interoffice envelope* is analogous to a *transport-layer segment*—it contains header information (Bob's name and department number) and it encapsulates the application-layer message (the memo). When the sending branch-office mailroom receives the interoffice envelope, it puts the interoffice envelope inside yet another envelope, which is suitable for sending through the public postal service. The sending mailroom also writes the postal address of the sending and receiving branch offices on the postal envelope. Here, the *postal envelope* is analogous to the *datagram*—it encapsulates the transport-layer segment (the interoffice envelope), which encapsulates the original message (the memo). The postal service delivers the postal envelope to the receiving branch-office mailroom. There, the process of de-encapsulation is begun. The mailroom extracts the interoffice memo and forwards it to Bob. Finally, Bob opens the envelope and removes the memo.

The process of encapsulation can be more complex than that described above. For example, a large message may be divided into multiple transport-layer segments (which might themselves each be divided into multiple network-layer datagrams). At the receiving end, such a segment must then be reconstructed from its constituent datagrams.

1.6 Networks Under Attack

The Internet has become mission critical for many institutions today, including large and small companies, universities, and government agencies. Many individuals also rely on the Internet for many of their professional, social, and personal activities. But behind all this utility and excitement, there is a dark side, a side where "bad guys" attempt to wreak havoc in our daily lives by damaging our Internet-connected computers, violating our privacy, and rendering inoperable the Internet services on which we depend [Skoudis 2006].

The field of network security is about how the bad guys can attack computer networks and about how we, soon-to-be experts in computer networking, can defend networks against those attacks, or better yet, design new architectures that are immune to such attacks in the first place. Given the frequency and variety of existing attacks as well as the threat of new and more destructive future attacks, network security has become a central topic in the field of computer networking in recent years. One of the features of this fourth edition of this textbook is that it brings network security issues

DELL - Exhibit 1008 - Page 78

to the forefront. We'll begin our foray into network security in this section, where we'll briefly describe some of the more prevalent and damaging attacks in today's Internet. Then, as we cover the various computer networking technologies and protocols in greater detail in subsequent chapters, we'll consider the various security-related issues associated with those technologies and protocols. Finally, in Chapter 8, armed with our newly acquired expertise in computer networking and Internet protocols, we'll study in-depth how computer networks can be defended against attacks, or designed and operated to make such attacks impossible in the first place.

Since we don't yet have expertise in computer networking and Internet protocols, we'll begin here by surveying some of today's more prevalent security-related problems. This will whet our appetite for more substantial discussions in the upcoming chapters. So we begin here by simply asking, what can go wrong? How are computer networks vulnerable? What are some of the more prevalent types of attacks today?

The bad guys can put malware into your host via the Internet

We attach devices to the Internet because we want to receive/send data from/to the Internet. This includes all kinds of good stuff, including Web pages, e-mail messages, MP3s, telephone calls, live video, search engine results, and so on. But, unfortunately, along with all that good stuff comes malicious stuff—collectively known as **malware**—that can also enter and infect our devices. Once malware infects our device it can do all kinds of devious things, including deleting our files; installing spyware that collects our private information, such as social security numbers, passwords, and keystrokes, and then sends this (over the Internet, of course!) back to the bad guys. Our compromised host may also be enrolled in a network of thousands of similarly compromised devices, collectively known as a **botnet**, which the bad guys control and leverage for spam e-mail distribution or distributed denial-of-service attacks (soon to be discussed) against targeted hosts.

Much of the malware out there today is **self-replicating**: once it infects one host, from that host it seeks entry into other hosts over the Internet, and from the newly infected hosts, it seeks entry into yet more hosts. In this manner, self-replicating malware can spread exponentially fast. For example, the number of devices infected by the 2003 Saphire/Slammer worm doubled every 8.5 seconds in the first few minutes after its outbreak, infecting more than 90 percent of vulnerable hosts within 10 minutes [Moore 2003]. Malware can spread in the form of a virus, a worm, or a Trojan horse [Skoudis 2004]. **Viruses** are malware that require some form of user interaction to infect the user's device. The classic example is an e-mail attachment containing malicious executable code. If a user receives and opens such an attachment, the user inadvertently runs the malware on the device. Typically, such e-mail viruses are self-replicating: once executed, the virus may send an identical message with an identical malicious attachment to, for example, every recipient in the user's address book. **Worms** (like the Slammer worm) are malware that can enter a device without any explicit user interaction. For example, a user may be

DELL - Exhibit 1008 - Page 79

running a vulnerable network application to which an attacker can send malware. In some cases, without any user intervention, the application may accept the malware from the Internet and run it, creating a worm. The worm in the newly infected device then scans the Internet, searching for other hosts running the same vulnerable network application. When it finds other vulnerable hosts, it sends a copy of itself to those hosts. Finally, a **Trojan horse** is malware that is a hidden part of some otherwise useful software. Today, malware, is pervasive and costly. For example, it has been estimated that the financial impact of viruses alone is over $14 billion in 2005 [Malware 2006]. As you work through this textbook, we encourage you to think about the following question: What can computer network designers do to defend Internet-attached devices from malware attacks?

The bad guys can attack servers and network infrastructure

A broad class of security threats can be classified as **denial-of-service (DoS) attacks**. As the name suggests, a DoS attack renders a network, host, or other piece of infrastructure unusable by legitimate users. Web servers, e-mail servers, DNS servers (discussed in Chapter 2), and institutional networks can all be subject to DoS attacks. Internet DoS attacks are extremely common, with thousands of DoS attacks occurring every year [Moore 2001; Mirkovic 2005]. Most Internet DoS attacks fall into one of three categories:

- *Vulnerability attack.* This involves sending a few well-crafted messages to a vulnerable application or operating system running on a targeted host. If the right sequence of packets is sent to a vulnerable application or operating system, the service can stop or, worse, the host can crash.
- *Bandwidth flooding.* The attacker sends a deluge of packets to the targeted host—so many packets that the target's access link becomes clogged, preventing legitimate packets from reaching the server.
- *Connection flooding.* The attacker establishes a large number of half-open or fully open TCP connections (TCP connections are discussed in Chapter 3) at the target host. The host can become so bogged down with these bogus connections that it stops accepting legitimate connections.

Let's now explore the bandwidth-flooding attack in more detail. Recalling our delay and loss analysis discussion in section 1.4.2, it's evident that if the server has an access rate of R bps, then the attacker will need to send traffic at a rate of approximately R bps to cause damage. If R is very large, a single attack source may not be able to generate enough traffic to harm the server. Furthermore, if all the traffic emanates from a single source, an upstream router may be able to detect the attack and block all traffic from that source before the traffic gets near the server. In a **distributed DoS (DDoS)** attack, illustrated in Figure 1.21, the attacker controls multiple sources

DELL - Exhibit 1008 - Page 80

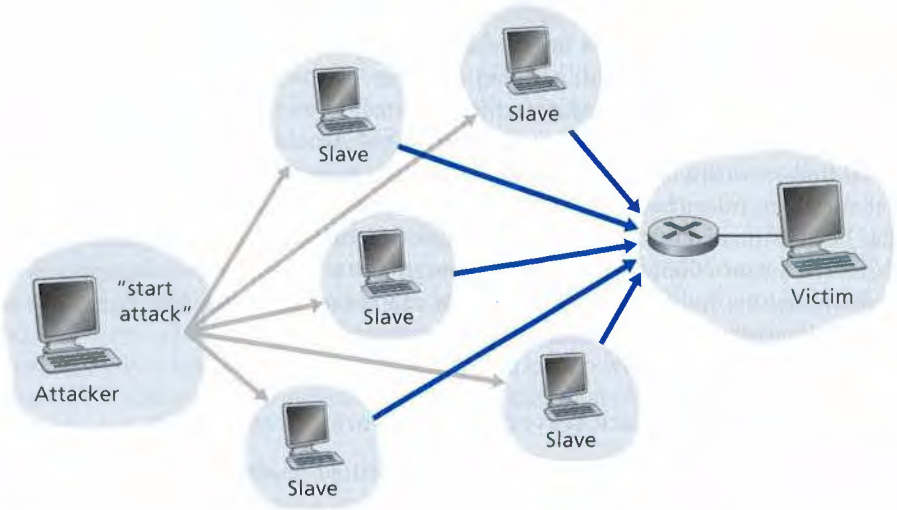


Figure 1.21 ♦ A distributed denial-of-service attack

and has each source blast traffic at the target. With this approach, the aggregate traffic rate across all the controlled sources needs to be approximately R to cripple the service. DDoS attacks leveraging botnets with thousands of comprised hosts are a common occurrence today [Mirkovic 2005]. DDos attacks are much harder to detect and defend against than a DoS attack from a single host.

We encourage you to consider the following question as you work your way through this book: What can computer network designers do to defend against DoS attacks? We will see that different defenses are needed for the three types of DoS attacks.

The bad guys can sniff packets

Many users today access the Internet via wireless devices, such as WiFi-connected laptops or handheld devices with cellular Internet connections (covered in Chapter 6). While ubiquitous Internet access is extremely convenient and enables marvelous new applications for mobile users, it also creates a major security vulnerability—by placing a passive receiver in the vicinity of the wireless transmitter, that receiver can obtain a copy of every packet that is transmitted! These packets can contain all kinds of sensitive information, including passwords, social security numbers, trade secrets, and private personal messages. A passive receiver that records a copy of every packet that flies by is called a **packet sniffer**.

Sniffers can be deployed in wired environments as well. In wired broadcast environments, as in many Ethernet LANs, a packet sniffer can obtain copies of

DELL - Exhibit 1008 - Page 81

all packets sent over the LAN. As described in Section 1.2, cable access technologies also broadcast packets and are thus vulnerable to sniffing. Furthermore, a bad guy who gains access to an institution's access router or access link to the Internet may be able to plant a sniffer that makes a copy of every packet going to/from the organization. Sniffed packets can then be analyzed offline for sensitive information.

Packet-sniffing software is freely available at various Web sites and as commercial products. Professors teaching a networking course have been known to assign lab exercises that involve writing a packet-sniffing and application-layer data reconstruction program. Indeed, the Ethereal [Ethereal 2007] labs associated with this text (see the introductory Ethereal lab at the end of this chapter) use exactly such a packet sniffer!

Because packet sniffers are passive—that is, they do not inject packets into the channel—they are difficult to detect. So, when we send packets into a wireless channel, we must accept the possibility that some bad guy may be recording copies of our packets. As you have may have guessed, some of the best defenses against packet sniffing involve cryptography. We will examine cryptography as it applies to network security in Chapter 8.

The bad guys can masquerade as someone you trust

It is surprisingly easy (*you* will have the knowledge to do so shortly as you proceed through this text!) to create a packet with an arbitrary source address, packet content, and destination address and then transmit this hand-crafted packet into the Internet, which will dutifully forward the packet to its destination. Imagine the unsuspecting receiver (say an Internet router) who receives such a packet, takes the (false) source address as being truthful, and then performs some command embedded in the packet's contents (say modifies its forwarding table). The ability to inject packets into the Internet with a false source address is known as **IP spoofing**, and is but one of many ways in which one user can masquerade as another user.

To solve this problem, we will need *end-point authentication*, that is, a mechanism that will allow us to determine with certainty if a message originates from where we think it does. Once again, we encourage you to think about how this can be done for network applications and protocols as you progress through the chapters of this book. We will explore mechanisms for end-point authentication in Chapter 8.

The bad guys can modify or delete messages

We end this brief survey of network attacks by describing **man-in-the-middle attacks**. In this class of attacks, the bad guy is inserted into the communication path between two communicating entities. Let's refer to the communicating entities as Alice and Bob, which might be actual human beings or might be network entities such as two routers or two e-mail servers. The bad guy could be, for example, a compromised router in the communication path, or a software module residing on one of the end hosts at a lower layer in the protocol stack. In the man-in-the-middle attack,

DELL - Exhibit 1008 - Page 82

the bad guy not only has the ability to sniff all packets that pass between Bob and Alice, but can also inject, modify, or delete packets. In the jargon of network security, a man-in-the-middle attack can compromise the *integrity* of the data sent between Alice and Bob. As we will see in Chapter 8, mechanisms that provide secrecy (protection against sniffing) and end-point authentication (allowing the receiver to verify with certainty the originator of the message) do not necessarily provide data integrity. So we will need yet another set of techniques to provide data integrity.

In closing this section, it's worth considering how the Internet got to be such an insecure place in the first place. The answer, in essence, is that the Internet was originally designed to be that way, based on the model of "a group of mutually trusting users attached to a transparent network" [Blumenthal 2001]—a model in which (by definition) there is no need for security. Many aspects of the original Internet architecture deeply reflect this notion of mutual trust. For example, the ability for one user to send a packet to any other user is the default rather than a requested/granted capability, and user identity is taken at declared face value, rather than being authenticated by default.

But today's Internet certainly does not involve "mutually trusting users." Nonetheless, today's users still need to communicate when they don't necessarily trust each other, may wish to communicate anonymously, may communicate indirectly through third parties (e.g., Web caches, which we'll study in Chapter 2, or mobility-assisting agents, which we'll study in Chapter 6), and may distrust the hardware, software, and even the air through which they communicate. We now have many security-related challenges before us as we progress through this book: we should seek defenses against sniffing, end-point masquerading, man-in-the-middle attacks, DDoS attacks, malware, and more. We should keep in mind that communication among mutually trusted users is the exception rather than the rule. Welcome to the world of modern computer networking!

1.7 History of Computer Networking and the Internet

Sections 1.1 through 1.6 presented an overview of the technology of computer networking and the Internet. You should know enough now to impress your family and friends! However, if you really want to be a big hit at the next cocktail party, you should sprinkle your discourse with tidbits about the fascinating history of the Internet [Segaller 1998].

1.7.1 The Development of Packet Switching: 1961–1972

The field of computer networking and today's Internet trace their beginnings back to the early 1960s, when the telephone network was the world's dominant

DELL - Exhibit 1008 - Page 83

communication network. Recall from Section 1.3 that the telephone network uses circuit switching to transmit information from a sender to a receiver—an appropriate choice given that voice is transmitted at a constant rate between sender and receiver. Given the increasing importance (and great expense) of computers in the early 1960s and the advent of timeshared computers, it was perhaps natural (at least with perfect hindsight!) to consider the question of how to hook computers together so that they could be shared among geographically distributed users. The traffic generated by such users was likely to be *bursty*—intervals of activity, such as the sending of a command to a remote computer, followed by periods of inactivity while waiting for a reply or while contemplating the received response.

Three research groups around the world, each unaware of the others' work [Leiner 1998], began inventing packet switching as an efficient and robust alternative to circuit switching. The first published work on packet-switching techniques was that of Leonard Kleinrock [Kleinrock 1961; Kleinrock 1964], then a graduate student at MIT. Using queuing theory, Kleinrock's work elegantly demonstrated the effectiveness of the packet-switching approach for bursty traffic sources. In 1964, Paul Baran [Baran 1964] at the Rand Institute had begun investigating the use of packet switching for secure voice over military networks, and at the National Physical Laboratory in England, Donald Davies and Roger Scantlebury were also developing their ideas on packet switching.

The work at MIT, Rand, and the NPL laid the foundations for today's Internet. But the Internet also has a long history of a let's-build-it-and-demonstrate-it attitude that also dates back to the 1960s. J. C. R. Licklider [DEC 1990] and Lawrence Roberts, both colleagues of Kleinrock's at MIT, went on to lead the computer science program at the Advanced Research Projects Agency (ARPA) in the United States. Roberts published an overall plan for the ARPAnet [Roberts 1967], the first packet-switched computer network and a direct ancestor of today's public Internet. The early packet switches were known as **interface message processors (IMPs),** and the contract to build these switches was awarded to the BBN company. On Labor Day in 1969, the first IMP was installed at UCLA under Kleinrock's supervision, and three additional IMPs were installed shortly thereafter at the Stanford Research Institute (SRI), UC Santa Barbara, and the University of Utah (Figure 1.22). The fledgling precursor to the Internet was four nodes large by the end of 1969. Kleinrock recalls the very first use of the network to perform a remote login from UCLA to SRI, crashing the system [Kleinrock 2004].

By 1972, ARPAnet had grown to approximately 15 nodes and was given its first public demonstration by Robert Kahn at the 1972 International Conference on Computer Communications. The first host-to-host protocol between ARPAnet end systems, known as the network-control protocol (NCP), was completed [RFC 001]. With an end-to-end protocol available, applications could now be written. Ray Tomlinson at BBN wrote the first e-mail program in 1972.

DELL - Exhibit 1008 - Page 84



Figure 1.22 ♦ An early interface message processor (IMP) and L. Kleinrock (Mark J. Terrill, AP/Wide World Photos)

1.7.2 Proprietary Networks and Internetworking: 1972–1980

The initial ARPAnet was a single, closed network. In order to communicate with an ARPAnet host, one had to be actually attached to another ARPAnet IMP. In the early to mid-1970s, additional stand-alone packet-switching networks besides ARPAnet came into being:

- ALOHANet, a microwave network linking universities on the Hawaiian islands [Abramson 1970], as well as DARPA's packet-satellite [RFC 829] and packet-radio networks [Kahn 1978]

DELL - Exhibit 1008 - Page 85

- Telenet, a BBN commercial packet-switching network based on ARPAnet technology
- Cyclades, a French packet-switching network pioneered by Louis Pouzin [Think 2007]
- Time-sharing networks such as Tymnet and the GE Information Services network, among others, in the late 1960s and early 1970s [Schwartz 1977]
- IBM's SNA (1969–1974), which paralleled the ARPAnet work [Schwartz 1977]

The number of networks was growing. With perfect hindsight we can see that the time was ripe for developing an encompassing architecture for connecting networks together. Pioneering work on interconnecting networks (under the sponsorship of the Defense Advanced Research Projects Agency (DARPA)), in essence creating a *network of networks,* was done by Vinton Cerf and Robert Kahn [Cerf 1974]; the term *internetting* was coined to describe this work.

These architectural principles were embodied in TCP. The early versions of TCP, however, were quite different from today's TCP. The early versions of TCP combined a reliable in-sequence delivery of data via end-system retransmission (still part of today's TCP) with forwarding functions (which today are performed by IP). Early experimentation with TCP, combined with the recognition of the importance of an unreliable, non-flow-controlled, end-to-end transport service for applications such as packetized voice, led to the separation of IP out of TCP and the development of the UDP protocol. The three key Internet protocols that we see today—TCP, UDP, and IP—were conceptually in place by the end of the 1970s.

In addition to the DARPA Internet-related research, many other important networking activities were underway. In Hawaii, Norman Abramson was developing ALOHAnet, a packet-based radio network that allowed multiple remote sites on the Hawaiian Islands to communicate with each other. The ALOHA protocol [Abramson 1970] was the first multiple-access protocol, allowing geographically distributed users to share a single broadcast communication medium (a radio frequency). Metcalfe and Boggs built on Abramson's multiple-access protocol work when they developed the Ethernet protocol [Metcalfe 1976] for wire-based shared broadcast networks; see Figure 1.23. Interestingly, Metcalfe and Boggs' Ethernet protocol was motivated by the need to connect multiple PCs, printers, and shared disks [Perkins 1994]. Twenty-five years ago, well before the PC revolution and the explosion of networks, Metcalfe and Boggs were laying the foundation for today's PC LANs. Ethernet technology represented an important step for internetworking as well. Each Ethernet local area network was itself a network, and as the number of LANs proliferated, the need to internetwork these LANs together became increasingly important. We'll discuss Ethernet, ALOHA, and other LAN technologies in detail in Chapter 5.

DELL - Exhibit 1008 - Page 86

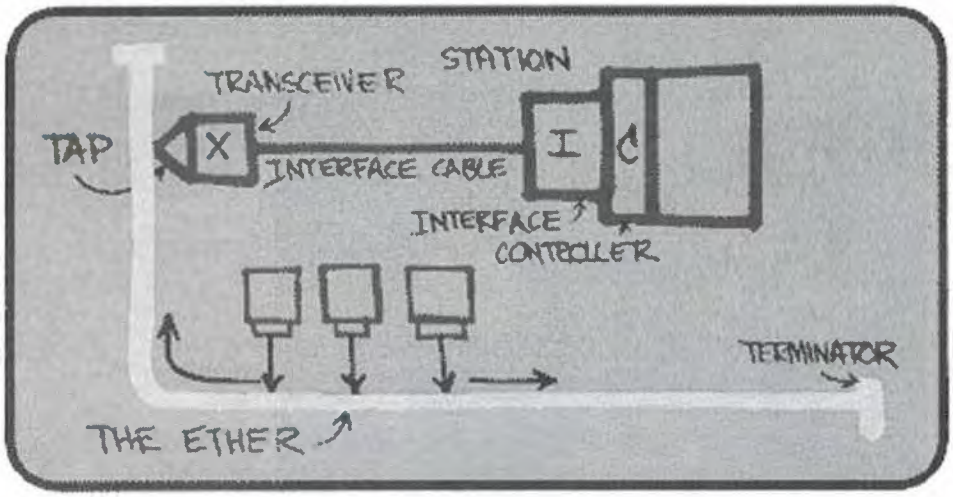


Figure 1.23 ♦ Metcalfe's original conception of the Ethernet

1.7.3 A Proliferation of Networks: 1980–1990

By the end of the 1970s, approximately two hundred hosts were connected to the ARPAnet. By the end of the 1980s the number of hosts connected to the public Internet, a confederation of networks looking much like today's Internet, would reach a hundred thousand. The 1980s would be a time of tremendous growth.

Much of that growth resulted from several distinct efforts to create computer networks linking universities together. BITNET provided e-mail and file transfers among several universities in the Northeast. CSNET (computer science network) was formed to link university researchers who did not have access to ARPAnet. In 1986, NSFNET was created to provide access to NSF-sponsored supercomputing centers. Starting with an initial backbone speed of 56 kbps, NSFNET's backbone would be running at 1.5 Mbps by the end of the decade and would serve as a primary backbone linking regional networks.

In the ARPAnet community, many of the final pieces of today's Internet architecture were falling into place. January 1, 1983 saw the official deployment of TCP/IP as the new standard host protocol for ARPAnet (replacing the NCP protocol). The transition [RFC 801] from NCP to TCP/IP was a flag day event—all hosts were required to transfer over to TCP/IP as of that day. In the late 1980s, important extensions were made to TCP to implement host-based congestion control [Jacobson 1988]. The DNS, used to map between a human-readable Internet name (for example, gaia.cs.umass.edu) and its 32-bit IP address, was also developed [RFC 1034].

Paralleling this development of the ARPAnet (which was for the most part a US effort), in the early 1980s the French launched the Minitel project, an ambitious

DELL - Exhibit 1008 - Page 87

plan to bring data networking into everyone's home. Sponsored by the French government, the Minitel system consisted of a public packet-switched network (based on the X.25 protocol suite), Minitel servers, and inexpensive terminals with built-in low-speed modems. The Minitel became a huge success in 1984 when the French government gave away a free Minitel terminal to each French household that wanted one. Minitel sites included free sites—such as a telephone directory site—as well as private sites, which collected a usage-based fee from each user. At its peak in the mid 1990s, it offered more than 20,000 services, ranging from home banking to specialized research databases. It was used by over 20 percent of France's population, generated more than $1 billion in revenue each year, and created 10,000 jobs. The Minitel was in a large proportion of French homes 10 years before most Americans had ever heard of the Internet.

1.7.4 The Internet Explosion: The 1990s

The 1990s were ushered in with a number of events that symbolized the continued evolution and the soon-to-arrive commercialization of the Internet. ARPAnet, the progenitor of the Internet, ceased to exist. MILNET and the Defense Data Network had grown in the 1980s to carry most of the US Department of Defense–related traffic and NSFNET had begun to serve as a backbone network connecting regional networks in the United States and national networks overseas. In 1991, NSFNET lifted its restrictions on the use of NSFNET for commercial purposes. NSFNET itself would be decommissioned in 1995, with Internet backbone traffic being carried by commercial Internet Service Providers.

The main event of the 1990s, however, was to be the emergence of the World Wide Web application, which brought the Internet into the homes and businesses of millions of people worldwide. The Web served as a platform for enabling and deploying hundreds of new applications, that we take for granted today. For a brief history of the early days of the Web, see [W3C 1995].

The Web was invented at CERN by Tim Berners-Lee between 1989 and 1991 [Berners-Lee 1989], based on ideas originating in earlier work on hypertext from the 1940s by Vannevar Bush [Bush 1945] and since the 1960s by Ted Nelson [Xanadu 2007]. Berners-Lee and his associates developed initial versions of HTML, HTTP, a Web server, and a browser—the four key components of the Web. Around the end of 1992 there were about two hundred Web servers in operation, this collection of servers being just a harbinger of what was about to come. At about this time several researchers were developing Web browsers with GUI interfaces, including Marc Andreessen, who led the development of the popular GUI browser Mosaic. In 1994 Marc Andreessen and Jim Clark formed Mosaic Communications, which later became Netscape Communications Corporation [Cusumano 1998; Quittner 1998]. By 1995, university students were using Mosaic and Netscape browsers to surf the Web on a daily basis. At about this time companies—big and small—began to operate Web servers and transact commerce over the Web. In 1996, Microsoft started to

DELL - Exhibit 1008 - Page 88

make browsers, which started the browser war between Netscape and Microsoft, which Microsoft won a few years later [Cusumano 1998].

The second half of the 1990s was a period of tremendous growth and innovation for the Internet, with major corporations and thousands of startups creating Internet products and services. Internet e-mail continued to evolve with feature-rich mail readers providing address books, attachments, hot links, and multimedia transport. By the end of the millennium the Internet was supporting hundreds of popular applications, including four killer applications:

- E-mail, including attachments and Web-accessible e-mail
- The Web, including Web browsing and Internet commerce
- Instant messaging, with contact lists, pioneered by ICQ
- Peer-to-peer file sharing of MP3s, pioneered by Napster

Interestingly, the first two killer applications came from the research community, whereas the last two were created by a few young entrepreneurs.

The period from 1995 to 2001 was a roller-coaster ride for the Internet in the financial markets. Before they were even profitable, hundreds of Internet startups made initial public offerings and started to be traded in a stock market. Many companies were valued in the billions of dollars without having any significant revenue streams. The Internet stocks collapsed in 2000–2001, and many startups shut down. Nevertheless, a number of companies emerged as big winners in the Internet space, including Microsoft, Cisco, Yahoo, e-Bay, Google, and Amazon.

1.7.5 Recent Developments

Innovation in computer networking continues at a rapid pace. Advances are being made on all fronts, including deployment of new applications, content distribution, Internet telephony, higher transmission speeds in LANs, and faster routers. But three developments merit special attention: a proliferation of high-speed access networking (including wireless access), security, and P2P networking.

As discussed in Section 1.2, increasing penetration of broadband residential Internet access via cable modem and DSL set the stage for a wealth of new multimedia applications, including voice and video over IP [Skype 2007], video sharing [YouTube 2007], and television over IP [PPLive 2007]. The increasing ubiquity of high-speed (11 Mbps and higher) public WiFi networks and medium-speed (hundreds of kbps) Internet access via cellular telephony networks is not only making it possible to remain constantly connected, but also enabling an exciting new set of location-specific services. We'll cover wireless networks and mobility in Chapter 6.

Following a series of denial-of-service attacks on prominent Web servers in the late 1990s and the proliferation of worm attacks (e.g., the Blaster worm), network security has become an immensely important topic. These attacks have resulted in

DELL - Exhibit 1008 - Page 89

the development of intrusion detection systems to provide early warning of an attack, and the use of firewalls to filter out unwanted traffic before it enters the network. We'll cover a number of important security-related topics in Chapter 8.

The last innovation of which we take special note is P2P networking. A P2P networking application exploits the resources in users' computers—storage, content, CPU cycles, and human presence—and has significant autonomy from central servers. Typically, the users' computers (i.e., the peers) have intermittent connectivity. There have been numerous P2P success stories in the past few years, including P2P file sharing (Napster, Kazaa, Gnutella, eDonkey, LimeWire, and so on), file distribution (BitTorrent), Voice over IP (Skype), and IPTV (PPLive, ppStream). We'll discuss many of these P2P applications in Chapter 2.

1.8 Summary

In this chapter we've covered a tremendous amount of material! We've looked at the various pieces of hardware and software that make up the Internet in particular and computer networks in general. We started at the edge of the network, looking at end systems and applications, and at the transport service provided to the applications running on the end systems. We also looked at the link-layer technologies and physical media typically found in the access network. We then dove deeper inside the network, into the network core, identifying packet switching and circuit switching as the two basic approaches for transporting data through a telecommunication network, and we examined the strengths and weaknesses of each approach. We also examined the structure of the global Internet, learning that the Internet is a network of networks. We saw that the Internet's hierarchical structure, consisting of higher- and lower-tier ISPs, has allowed it to scale to include thousands of networks.

In the second part of this introductory chapter, we examined several topics central to the field of computer networking. We first examined the causes of delay, throughput and packet loss in a packet-switched network. We developed simple quantitative models for transmission, propagation, and queuing delays as well as for throughput; we'll make extensive use of these delay models in the homework problems throughout this book. Next we examined protocol layering and service models, key architectural principles in networking that we will also refer back to throughout this book. We also surveyed some of the more prevalent security attacks in the Internet day. We finished our introduction to networking with a brief history of computer networking. The first chapter in itself constitutes a mini-course in computer networking.

So, we have indeed covered a tremendous amount of ground in this first chapter! If you're a bit overwhelmed, don't worry. In the following chapters we'll revisit all of these ideas, covering them in much more detail (that's a promise, not a threat!). At this point, we hope you leave this chapter with a still-developing intuition for the pieces that make up a network, a still-developing command of the

DELL - Exhibit 1008 - Page 90

vocabulary of networking (don't be shy about referring back to this chapter), and an ever-growing desire to learn more about networking. That's the task ahead of us for the rest of this book.

Road-Mapping This Book

Before starting any trip, you should always glance at a road map in order to become familiar with the major roads and junctures that lie ahead. For the trip we are about to embark on, the ultimate destination is a deep understanding of the how, what, and why of computer networks. Our road map is the sequence of chapters of this book:

1. Computer Networks and the Internet
2. Application Layer
3. Transport Layer
4. Network Layer
5. Link Layer and Local Area Networks
6. Wireless and Mobility
7. Multimedia Networking
8. Security in Computer Networks
9. Network Management

Chapters 2 through 5 are the four core chapters of this book. You should notice that these chapters are organized around the top four layers of the five-layer Internet protocol stack, one chapter for each layer. Further note that our journey will begin at the top of the Internet protocol stack, namely, the application layer, and will work its way downward. The rationale behind this top-down journey is that once we understand the applications, we can understand the network services needed to support these applications. We can then, in turn, examine the various ways in which such services might be implemented by a network architecture. Covering applications early thus provides motivation for the remainder of the text.

The second half of the book—Chapters 6 through 9—zooms in on four enormously important (and somewhat independent) topics in modern computer networking. In Chapter 6, we examine wireless and mobility, including wireless LANs (including WiFi, Wi MAX, and Bluetooth), Cellular telephony networks (including GSM), and mobility (in both IP and GSM networks). In Chapter 7 (Multimedia Networking) we examine audio and video applications such as Internet phone, video conferencing, and streaming of stored media. We also look at how a packet-switched network can be designed to provide consistent quality of service to audio and video applications. In Chapter 8 (Security in Computer Networks), we first look at the underpinnings of encryption and network security, and then we examine how the basic theory is being applied in a broad range of Internet contexts. The last chapter (Network Management) examines the key issues in network management as well as the primary Internet protocols used for network management.

DELL - Exhibit 1008 - Page 91

CHAPTER 4

The Network Layer

We learned in the previous chapter that the transport layer provides various forms of process-to-process communication by relying on the network layer's host-to-host communication service. We also learned that the transport layer does so without any knowledge about how the network layer actually implements this service. So perhaps you're now wondering, what's under the hood of the host-to-host communication service, what makes it tick?

In this chapter we'll learn exactly how the network layer implements the host-to-host communication service. We'll see that unlike the transport layer, there is a piece of the network layer in each and every host and router in the network. Because of this, network-layer protocols are among the most challenging (and therefore among the most interesting!) in the protocol stack.

The network layer is also one of the most complex layers in the protocol stack, and so we'll have a lot of ground to cover here. We'll begin our study with an overview of the network layer and the services it can provide. We'll then revisit the two broad approaches towards structuring network-layer packet delivery—the datagram and the virtual-circuit model—that we first encountered back in Chapter 1, and see the fundamental role that addressing plays in delivering a packet to its destination host.

In this chapter, we'll make an important distinction between the **forwarding** and **routing** functions of the network layer. Forwarding involves the transfer of a

DELL - Exhibit 1008 - Page 92

packet from an incoming link to an outgoing link within a *single* router. Routing involves *all* of a network's routers, whose collective interactions via routing protocols determine the paths that packets take on their trips from source to destination node. Keeping this distinction in mind as you progress through this chapter will help you place many of the topics covered in an appropriate context.

In order to deepen our understanding of packet forwarding, we'll look "inside" a router—at its hardware architecture and organization. We'll then look at packet forwarding in the Internet, along with the celebrated Internet Protocol (IP). We'll investigate network-layer addressing and the IPv4 datagram format. We'll then explore network address translation (NAT), datagram fragmentation, the Internet Control Message Protocol (ICMP), and IPv6.

We'll then turn our attention to the network layer's routing function. We'll see that the job of a routing algorithm is to determine good paths (equivalently, routes) from senders to receivers. We'll first study the theory of routing algorithms, concentrating on the two most prevalent classes of algorithms: link-state and distance-vector algorithms. Since the complexity of routing algorithms grows considerably as the number of network routers increases, hierarchical routing approaches will also be of interest. We'll then see how theory is put into practice when we cover the Internet's intra-autonomous system routing protocols (RIP, OSPF, and IS-IS) and its inter-autonomous system routing protocol, BGP. We'll close this chapter with a discussion of broadcast and multicast routing.

In summary, this chapter has three major parts. The first part, Sections 4.1 and 4.2, covers network-layer functions and services. The second part, Sections 4.3 and 4.4, covers forwarding. Finally, the third part, Sections 4.5 through 4.7, covers routing.

4.1 Introduction

Figure 4.1 shows a simple network with two hosts, H1 and H2, and several routers on the path between H1 and H2. Suppose that H1 is sending information to H2, and consider the role of the network layer in these hosts and in the intervening routers. The network layer in H1 takes segments from the transport layer in H1, encapsulates each segment into a datagram (that is, a network-layer packet), and then sends the datagrams to its nearby router, R1. At the receiving host, H2, the network layer receives the datagrams from its nearby router R2, extracts the transport-layer segments, and delivers the segments up to the transport layer at H2. The primary role of the routers is to forward datagrams from input links to output links. Note that the routers in Figure 4.1 are shown with a truncated protocol stack, that is, with no upper layers above the network layer, because (except for control purposes) routers do not run application- and transport-layer protocols such as those we examined in Chapters 2 and 3.

DELL - Exhibit 1008 - Page 93

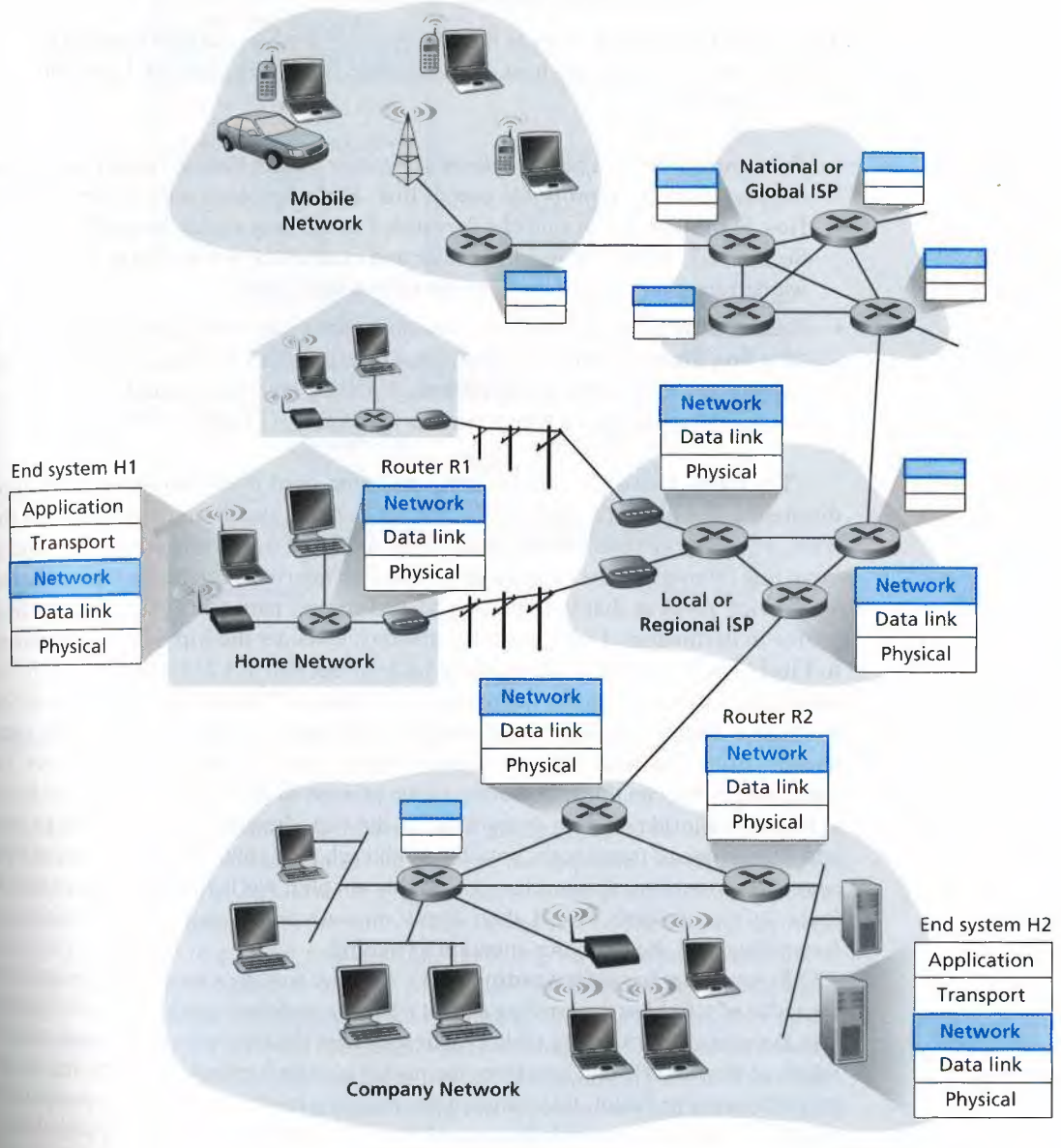


Figure 4.1 ♦ The network layer

DELL - Exhibit 1008 - Page 94

4.1.1 Forwarding and Routing

The role of the network layer is thus deceptively simple—to move packets from a sending host to a receiving host. To do so, two important network-layer functions can be identified:

- *Forwarding.* When a packet arrives at a router's input link, the router must move the packet to the appropriate output link. For example, a packet arriving from Host H1 to Router R1 must be forwarded to the next router on a path to H2. In Section 4.3, we'll look inside a router and examine how a packet is actually forwarded from an input link at a router to an output link.
- *Routing.* The network layer must determine the route or path taken by packets as they flow from a sender to a receiver. The algorithms that calculate these paths are referred to as **routing algorithms**. A routing algorithm would determine, for example, the path along which packets flow from H1 to H2.

The terms *forwarding* and *routing* are often used interchangeably by authors discussing the network layer. We'll use these terms much more precisely in this book. *Forwarding* refers to the router-local action of transferring a packet from an input link interface to the appropriate output link interface. *Routing* refers to the network-wide process that determines the end-to-end paths that packets take from source to destination. Using a driving analogy, consider the trip from Pennsylvania to Florida undertaken by our traveler back in Section 1.3.2. During this trip, our driver passes through many interchanges en route to Florida. We can think of forwarding as the process of getting through a single interchange: A car enters the interchange from one road and determines which road it should take to leave the interchange. We can think of routing as the process of planning the trip from Pennsylvania to Florida: Before embarking on the trip, the driver has consulted a map and chosen one of many paths possible, with each path consisting of a series of road segments connected at interchanges. In this chapter, we'll first examine network-layer service models. We'll then focus on networking layer topics related to forwarding, and then turn our attention to routing.

Every router has a **forwarding table**. A router forwards a packet by examining the value of a field in the arriving packet's header, and then using this value to index into the router's forwarding table. The result from the forwarding table indicates to which of the router's link interfaces the packet is to be forwarded. Depending on the network-layer protocol, this value in the packet's header could be the destination address of the packet or an indication of the connection to which the packet belongs. Figure 4.2 provides an example. In Figure 4.2, a packet with a header field value of 0111 arrives to a router. The router indexes into its forwarding table and determines that the output link interface for this packet is interface 2. The router then internally forwards the packet to interface 2. In Section 4.3 we'll look inside a router and examine the forwarding function in much greater detail.

DELL - Exhibit 1008 - Page 95

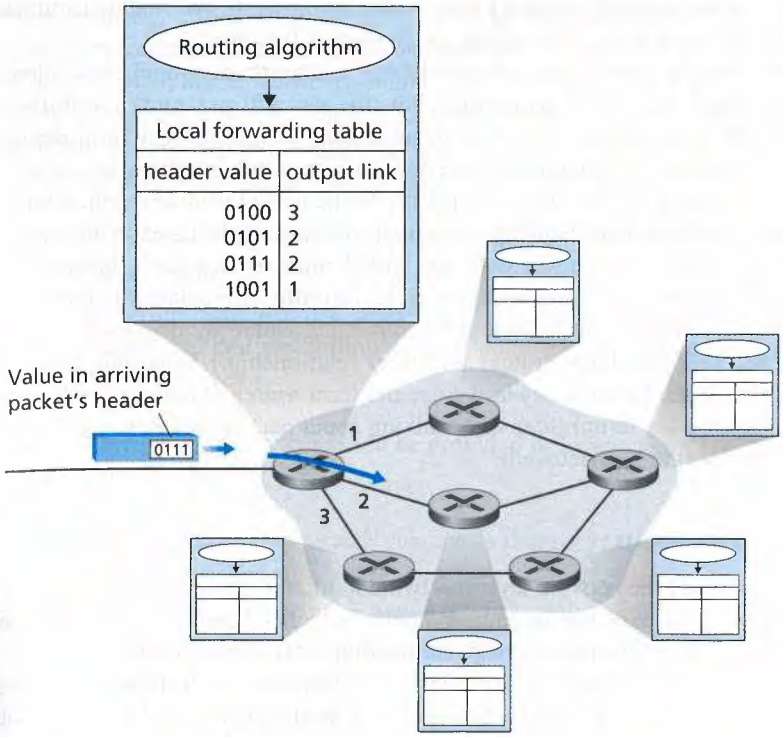


Figure 4.2 ♦ Routing algorithms determine values in forwarding tables.

You might now be wondering how the forwarding tables in the routers are configured. This is a crucial issue, one that exposes the important interplay between routing and forwarding. As shown in Figure 4.2, the routing algorithm determines the values that are inserted into the routers' forwarding tables. The routing algorithm may be centralized (e.g., with an algorithm executing on a central site and downloading routing information to each of the routers) or decentralized (i.e., with a piece of the distributed routing algorithm running in each router). In either case, a router receives routing protocol messages, which are used to configure its forwarding table. The distinct and different purposes of the forwarding and routing functions can be further illustrated by considering the hypothetical (and unrealistic, but technically feasible) case of a network in which all forwarding tables are configured directly by human network operators physically present at the routers. In this case, *no* routing protocols would be required! Of course, the human operators would need to interact with each other to ensure that the forwarding tables were configured in such a way that packets reached their intended destinations. It's also likely that human configuration would be more error-prone and much slower to respond to

DELL - Exhibit 1008 - Page 96

changes in the network topology than a routing protocol. We're thus fortunate that all networks have both a forwarding *and* a routing function!

While we're on the topic of terminology, it's worth mentioning two other terms that are often used interchangeably, but that we will use more carefully. We'll reserve the term *packet switch* to mean a general packet-switching device that transfers a packet from input link interface to output link interface, according to the value in a field in the header of the packet. Some packet switches, called **link-layer switches** (examined in Chapter 5), base their forwarding decision on the value in the link-layer field. Other packet switches, called **routers**, base their forwarding decision on the value in the network-layer field. (To fully appreciate this important distinction, you might want to review Section 1.5.2, where we discuss network-layer datagrams and link-layer frames and their relationship.) Since our focus in this chapter is on the network layer, we use the term *router* in place of *packet switch*. We'll even use the term *router* when talking about packet switches in virtual-circuit networks (soon to be discussed).

Connection Setup

We just said that the network layer has two important functions, forwarding and routing. But we'll soon see that in some computer networks there is actually a third important network-layer function, namely, **connection setup**. Recall from our study of TCP that a three-way handshake is required before data can flow from sender to receiver. This allows the sender and receiver to set up the needed state information (for example, sequence number and initial flow-control window size). In an analogous manner, some network-layer architectures—for example, ATM and frame-relay, but not the Internet—require the routers along the chosen path from source to destination to handshake with each other in order to set up state before network-layer data packets within a given source-to-destination connection can begin to flow. In the network layer, this process is referred to as *connection setup*. We'll examine connection setup in Section 4.2.

4.1.2 Network Service Models

Before delving into the network layer, let's take the broader view and consider the different types of service that might be offered by the network layer. When the transport layer at a sending host transmits a packet into the network (that is, passes it down to the network layer at the sending host), can the transport layer count on the network layer to deliver the packet to the destination? When multiple packets are sent, will they be delivered to the transport layer in the receiving host in the order in which they were sent? Will the amount of time between the sending of two sequential packet transmissions be the same as the amount of time between their reception? Will the network provide any feedback about congestion in the network? What is the abstract view (properties) of the channel connecting the transport layer in the sending and receiving hosts? The answers to these questions and others are determined

DELL - Exhibit 1008 - Page 97

by the service model provided by the network layer. The **network service model** defines the characteristics of end-to-end transport of packets between one edge of the network and the other, that is, between sending and receiving end systems.

Let's now consider some possible services that the network layer could provide. In the sending host, when the transport layer passes a packet to the network layer, specific services that could be provided by the network layer include:

- *Guaranteed delivery*. This service guarantees that the packet will eventually arrive at its destination.
- *Guaranteed delivery with bounded delay*. This service not only guarantees delivery of the packet, but delivery within a specified host-to-host delay bound (for example, within 100 msec).

Furthermore, the following services could be provided to a *flow of packets* between a given source and destination:

- *In-order packet delivery*. This service guarantees that packets arrive at the destination in the order that they were sent.
- *Guaranteed minimal bandwidth*. This network-layer service emulates the behavior of a transmission link of a specified bit rate (for example, 1 Mbps) between sending and receiving hosts (even though the actual end-to-end path may traverse several physical links). As long as the sending host transmits bits (as part of packets) at a rate below the specified bit rate, then no packet is lost and each packet arrives within a prespecified host-to-host delay (for example, within 40 msec).
- *Guaranteed maximum jitter*. This service guarantees that the amount of time between the transmission of two successive packets at the sender is equal to the amount of time between their receipt at the destination (or that this spacing changes by no more than some specified value).
- *Security services*. Using a secret session key known only by a source and destination host, the network layer in the source host could encrypt the payloads of all datagrams being sent to the destination host. The network layer in the destination host would then be responsible for decrypting the payloads. With such a service, confidentiality would be provided to all transport-layer segments (TCP and UDP) between the source and destination hosts. In addition to confidentiality, the network layer could provide data integrity and source authentication services.

This is only a partial list of services that a network layer could provide—there are countless variations possible.

The Internet's network layer provides a single service, known as **best-effort service**. From Table 4.1, it might appear that *best-effort service* is a euphemism for *no service at all*. With best-effort service, timing between packets is not guaranteed

DELL - Exhibit 1008 - Page 98

Network Architecture	Service Model	Bandwidth Guarantee	No-Loss Guarantee	Ordering	Timing	Congestion Indication
Internet	Best Effort	None	None	Any order possible	Not maintained	None
ATM	CBR	Guaranteed constant rate	Yes	In order	Maintained	Congestion will not occur
ATM	ABR	Guaranteed minimum	None	In order	Not maintained	Congestion indication provided

Table 4.1 ♦ Internet, ATM CBR, and ATM ABR service models

to be preserved, packets are not guaranteed to be received in the order in which they were sent, nor is the eventual delivery of transmitted packets guaranteed. Given this definition, a network that delivered *no* packets to the destination would satisfy the definition of best-effort delivery service. As we'll discuss shortly, however, there are sound reasons for such a minimalist network-layer service model. We'll cover additional, still-evolving, Internet service models in Chapter 7.

Other network architectures have defined and implemented service models that go beyond the Internet's best-effort service. For example, the ATM network architecture [MFA Forum 2007, Black 1995] provides for multiple service models, meaning that different connections can be provided with different classes of service within the same network. A discussion of how an ATM network provides such services is well beyond the scope of this book; our aim here is only to note that alternatives do exist to the Internet's best-effort model. Two of the more important ATM service models are constant bit rate and available bit rate service:

- ***Constant bit rate (CBR) ATM network service.*** This was the first ATM service model to be standardized, reflecting early interest by the telephone companies in ATM and the suitability of CBR service for carrying real-time, constant bit rate audio and video traffic. The goal of CBR service is conceptually simple—to provide a flow of packets (known as cells in ATM terminology) with a virtual pipe whose properties are the same as if a dedicated fixed-bandwidth transmission link existed between sending and receiving hosts. With CBR service, a flow of ATM cells is carried across the network in such a way that a cell's end-to-end delay, the variability in a cell's end-end delay (that is, the jitter), and the fraction of cells that are lost or delivered late are all guaranteed to be less than specified values. These values are agreed upon by the sending host and the ATM network when the CBR connection is first established.

DELL - Exhibit 1008 - Page 99

- ***Available bit rate (ABR) ATM network service.*** With the Internet offering so-called best-effort service, ATM's ABR might best be characterized as being a slightly-better-than-best-effort service. As with the Internet service model, cells may be lost under ABR service. Unlike in the Internet, however, cells cannot be reordered (although they may be lost), and a minimum cell transmission rate (MCR) is guaranteed to a connection using ABR service. If the network has enough free resources at a given time, a sender may also be able to send cells successfully at a higher rate than the MCR. Additionally, as we saw in Section 3.6, ATM ABR service can provide feedback to the sender (in terms of a congestion notification bit, or an explicit rate at which to send) that controls how the sender adjusts its rate between the MCR and an allowable peak cell rate.

4.2 Virtual Circuit and Datagram Networks

Recall from Chapter 3 that a transport layer can offer applications connectionless service or connection-oriented service. For example, the Internet's transport layer provides each application a choice between two services: UDP, a connectionless service; or TCP, a connection-oriented service. In a similar manner, a network layer can also provide connectionless service or connection service. Network-layer connection and connectionless services in many ways parallel transport-layer connection-oriented and connectionless services. For example, a network-layer connection service begins with handshaking between the source and destination hosts; and a network-layer connectionless service does not have any handshaking preliminaries.

Although the network-layer connection and connectionless services have some parallels with transport-layer connection-oriented and connectionless services, there are crucial differences:

- In the network layer these services are host-to-host services provided by the network layer to the transport layer. In the transport layer these services are process-to-process services provided by the transport layer to the application layer.
- In all major computer network architectures to date (Internet, ATM, frame relay, and so on), the network layer provides either a host-to-host connectionless service or a host-to-host connection service, but not both. Computer networks that provide only a connection service at the network layer are called **virtual-circuit (VC) networks**; computer networks that provide only a connectionless service at the network layer are called **datagram networks**.
- The implementations of connection-oriented service in the transport layer and the connection service in the network layer are fundamentally different. We saw in the previous chapter that the transport-layer connection-oriented service is implemented at the edge of the network in the end systems; we'll see shortly that

DELL - Exhibit 1008 - Page 100

the network-layer connection service is implemented in the routers in the network core as well as in the end systems.

Virtual-circuit and datagram networks are two fundamental classes of computer networks. They use very different information in making their forwarding decisions. Let's now take a closer look at their implementations.

4.2.1 Virtual-Circuit Networks

We've learned that the Internet is a datagram network. However, many alternative network architectures—including those of ATM and frame relay—are virtual-circuit networks and, therefore, use connections at the network layer. These network-layer connections are called **virtual circuits (VCs)**. Let's now consider how a VC service can be implemented in a computer network.

A VC consists of (1) a path (that is, a series of links and routers) between the source and destination hosts, (2) VC numbers, one number for each link along the path, and (3) entries in the forwarding table in each router along the path. A packet belonging to a virtual circuit will carry a VC number in its header. Because a virtual circuit may have a different VC number on each link, each intervening router must replace the VC number of each traversing packet with a new VC number. The new VC number is obtained from the forwarding table.

To illustrate the concept, consider the network shown in Figure 4.3. The numbers next to the links of R1 in Figure 4.3 are the link interface numbers. Suppose now that Host A requests that the network establish a VC between itself and Host B. Suppose also that the network chooses the path A-R1-R2-B and assigns VC numbers 12, 22, 32 to the three links in this path for this virtual circuit. In this case, when a packet in this VC leaves Host A, the value in the VC number field in the packet header is 12; when it leaves R1, the value is 22; and when it leaves R2, the value is 32.

How does the router determine the replacement VC number for a packet traversing the router? For a VC network, each router's forwarding table includes VC

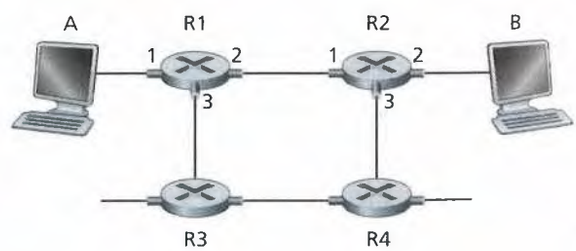


Figure 4.3 ♦ A simple virtual circuit network

DELL - Exhibit 1008 - Page 101

number translation; for example, the forwarding table in R1 might look something like this:

Incoming Interface	Incoming VC #	Outgoing Interface	Outgoing VC #
1	12	2	22
2	63	1	18
3	7	2	17
1	97	3	87
...	...	...	...

Whenever a new VC is established across a router, an entry is added to the forwarding table. Similarly, whenever a VC terminates, the appropriate entries in each table along its path are removed.

You might be wondering why a packet doesn't just keep the same VC number on each of the links along its route. The answer is twofold. First, replacing the number from link to link reduces the length of the VC field in the packet header. Second, and more importantly, VC setup is considerably simplified by permitting a different VC number at each link along the path of the VC. Specifically, with multiple VC numbers, each link in the path can choose a VC number independently of the VC numbers chosen at other links along the path. If a common VC number were required for all links along the path, the routers would have to exchange and process a substantial number of messages to agree on a common VC number (e.g., one that is not being used by any other existing VC at these routers) to be used for a connection.

In a VC network, the network's routers must maintain **connection state information** for the ongoing connections. Specifically, each time a new connection is established across a router, a new connection entry must be added to the router's forwarding table; and each time a connection is released, an entry must be removed from the table. Note that even if there is no VC-number translation, it is still necessary to maintain connection state information that associates VC numbers with output interface numbers. The issue of whether or not a router maintains connection state information for each ongoing connection is a crucial one—one that we'll return to repeatedly in this book.

There are three identifiable phases in a virtual circuit:

- *VC setup.* During the setup phase, the sending transport layer contacts the network layer, specifies the receiver's address, and waits for the network to set up the VC. The network layer determines the path between sender and receiver, that is, the series of links and routers through which all packets of the VC will travel. The network layer also determines the VC number for each link along the path. Finally, the network layer adds an entry in the forwarding table in each router

DELL - Exhibit 1008 - Page 102

along the path. During VC setup, the network layer may also reserve resources (for example, bandwidth) along the path of the VC.

- *Data transfer.* As shown in Figure 4.4, once the VC has been established, packets can begin to flow along the VC.
- *VC teardown.* This is initiated when the sender (or receiver) informs the network layer of its desire to terminate the VC. The network layer will then typically inform the end system on the other side of the network of the call termination and update the forwarding tables in each of the packet routers on the path to indicate that the VC no longer exists.

There is a subtle but important distinction between VC setup at the network layer and connection setup at the transport layer (for example, the TCP three-way handshake we studied in Chapter 3). Connection setup at the transport layer involves only the two end systems. During transport-layer connection setup, the two end systems alone determine the parameters (for example, initial sequence number and flow-control window size) of their transport-layer connection. Although the two end systems are aware of the transport-layer connection, the routers within the network are completely oblivious to it. On the other hand, with a VC network layer, *routers along the path between the two end systems are involved in VC setup, and each router is fully aware of all the VCs passing through it.*

The messages that the end systems send into the network to initiate or terminate a VC, and the messages passed between the routers to set up the VC (that is, to modify connection state in router tables) are known as **signaling messages**, and the protocols

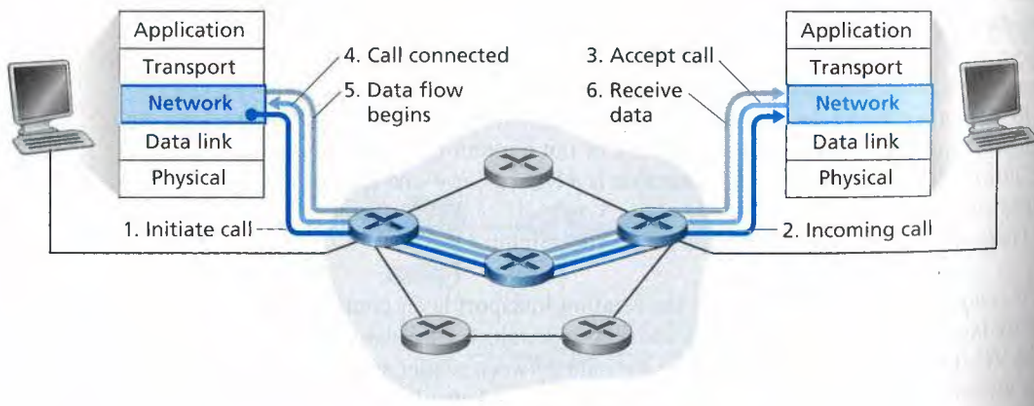


Figure 4.4 ♦ Virtual-circuit setup

DELL - Exhibit 1008 - Page 103

used to exchange these messages are often referred to as **signaling protocols**. VC setup is shown pictorially in Figure 4.4. We'll not cover VC signaling protocols in this book; see [Black 1997] for a general discussion of signaling in connection-oriented networks and [ITU-T Q.2931 1994] for the specification of ATM's Q.2931 signaling protocol.

4.2.2 Datagram Networks

In a **datagram network**, each time an end system wants to send a packet, it stamps the packet with the address of the destination end system and then pops the packet into the network. As shown in Figure 4.5, this is done without any VC setup. Routers in a datagram network do not maintain any state information about VCs (because there are no VCs!).

As a packet is transmitted from source to destination, it passes through a series of routers. Each of these routers uses the packet's destination address to forward the packet. Specifically, each router has a forwarding table that maps destination addresses to link interfaces; when a packet arrives at the router, the router uses the packet's destination address to look up the appropriate output link interface in the forwarding table. The router then intentionally forwards the packet to that output link interface.

To get some further insight into the lookup operation, let's look at a specific example. Suppose that all destination addresses are 32 bits (which just happens to be the length of the destination address in an IP datagram). A brute-force implementation of the forwarding table would have one entry for every possible destination address. Since there are more than 4 billion possible addresses, this option is totally out of the question—it would require a humongous forwarding table.

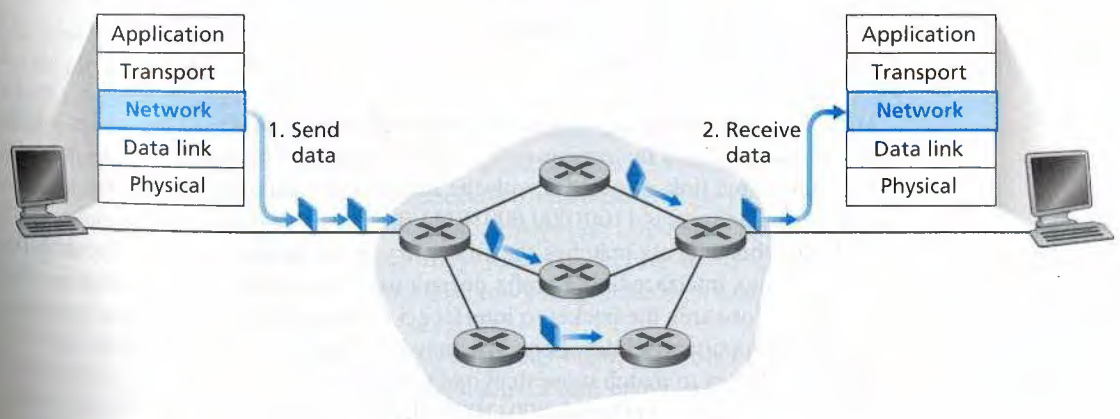


Figure 4.5 ♦ Datagram network

DELL - Exhibit 1008 - Page 104

Now let's further suppose that our router has four links, numbered 0 through 3, and that packets are to be forwarded to the link interfaces as follows:

Destination Address Range	Link Interface
11001000 00010111 00010000 00000000 through 11001000 00010111 00010111 11111111	0
11001000 00010111 00011000 00000000 through 11001000 00010111 00011000 11111111	1
11001000 00010111 00011001 00000000 through 11001000 00010111 00011111 11111111	2
otherwise	3

Clearly, for this example, it is not necessary to have 4 billion entries in the router's forwarding table. We could, for example, have the following forwarding table with just four entries:

Prefix Match	Link Interface
11001000 00010111 00010	0
11001000 00010111 00011000	1
11001000 00010111 00011	2
otherwise	3

With this style of forwarding table, the router matches a **prefix** of the packet's destination address with the entries in the table; if there's a match, the router forwards the packet to a link associated with the match. For example, suppose the packet's destination address is 11001000 00010111 00010110 10100001; because the 21-bit prefix of this address matches the first entry in the table, the router forwards the packet to link interface 0. If a prefix doesn't match any of the first three entries, then the router forwards the packet to interface 3. Although this sounds simple enough, there's an important subtlety here. You may have noticed that it is possible for a destination address to match more than one entry. For example, the first 24 bits of the address 11001000 00010111 00011000 10101010 match the second entry in the table, and the first 21 bits of the address match the third entry in the table. When there are multiple matches, the router uses the **longest prefix matching rule**; that

DELL - Exhibit 1008 - Page 105

is, it finds the longest matching entry in the table and forwards the packet to the link interface associated with the longest prefix match.

Of course, for longest prefix matching to be effective, each output link interface should be responsible for forwarding large blocks of contiguous destination addresses. We'll see in Section 4.4 that Internet addresses are typically assigned in a hierarchical fashion so that this contiguous property is prevalent in the forwarding tables of most routers. Nevertheless, there is some concern within the Internet research community that more and more holes are being punctured into the address space, causing the contiguous blocks to get smaller and smaller and the forwarding tables to get larger and larger. (See [Meng 2005], [RFC 3221], and the Principles in Practice discussion in Section 4.4.)

Although routers in datagram networks maintain no connection state information, they nevertheless maintain forwarding state information in their forwarding tables. However, the time scale at which this forwarding state information changes is relatively slow. Indeed, in a datagram network the forwarding tables are modified by the routing algorithms, which typically update a forwarding table every one-to-five minutes or so. In a VC network, a forwarding table in a router is modified whenever a new connection is set up through the router or whenever an existing connection through the router is torn down. This could easily happen at a microsecond timescale in a backbone, tier-1 router.

Because forwarding tables in datagram networks can be modified at any time, a series of packets sent from one end system to another may follow different paths through the network and may arrive out of order. [Paxson 1997] and [Jaiswal 2003] present interesting measurement studies of packet reordering and other phenomena in the public Internet.

4.2.3 Origins of VC and Datagram Networks

The evolution of datagram and VC networks reflects their origins. The notion of a virtual circuit as a central organizing principle has its roots in the telephony world, which uses real circuits. With call setup and per-call state being maintained at the routers within the network, a VC network is arguably more complex than a datagram network (although see [Molinero-Fernandez 2002] for an interesting comparison of the complexity of circuit- versus packet-switched networks). This, too, is in keeping with its telephony heritage. Telephone networks, by necessity, had their complexity within the network, since they were connecting dumb end-system devices such as rotary telephones. (For those too young to know, a rotary phone is an analog telephone with no buttons—only a dial.)

The Internet as a datagram network, on the other hand, grew out of the need to connect computers together. Given more sophisticated end-system devices, the Internet architects chose to make the network-layer service model as simple as possible. As we have already seen in Chapters 2 and 3, additional functionality (for example, in-order delivery, reliable data transfer, congestion control, and DNS name

DELL - Exhibit 1008 - Page 106

resolution) is then implemented at a higher layer, in the end systems. This inverts the model of the telephone network, with some interesting consequences:

- The resulting Internet network-layer service model, which makes minimal (no!) service guarantees (and hence imposes minimal requirements on the network layer), also makes it easier to interconnect networks that use very different link-layer technologies (for example, satellite, Ethernet, fiber, or radio) and have very different transmission rates and loss characteristics. We will address the interconnection of IP networks in detail in Section 4.4.
- As we saw in Chapter 2, applications such as e-mail, the Web, and even a network layer–centric service such as the DNS are implemented in hosts (servers) at the edge of the network. The ability to add a new service simply by attaching a host to the network and defining a new application-layer protocol (such as HTTP) has allowed new applications such as the Web to be deployed in the Internet in a remarkably short period of time.

As we'll see in Chapter 7, there is considerable debate in the Internet community about how the Internet's network-layer architecture should evolve in order to support real-time services such as multimedia. An interesting comparison of the VC-oriented ATM network architecture and a proposed next-generation Internet architecture is given in [Crowcroft 1995].

4.3 What's Inside a Router?

Now that we've seen an overview of the functions and services of the network layer, let's turn our attention to the network layer's **forwarding function**—the actual transfer of packets from a router's incoming links to the appropriate outgoing links. We already took a brief look at a few forwarding issues in Section 4.2, namely, addressing and longest prefix matching. In this section we'll look at specific router architectures for transferring packets from incoming links to outgoing links. Our coverage here is necessarily brief, as an entire course would be needed to cover router design in depth. Consequently, we'll make a special effort in this section to provide pointers to material that covers this topic in more depth. We mention here in passing that the words *forwarding* and *switching* are often used interchangeably by computer-networking researchers and practitioners; we'll use both terms in this textbook.

A high-level view of a generic router architecture is shown in Figure 4.6. Four components of a router can be identified:

- *Input ports.* The input port performs several functions. It performs the physical-layer functions (the leftmost box of the input port and the rightmost box of the output port in Figure 4.6) of terminating an incoming physical link to a router. It

DELL - Exhibit 1008 - Page 107

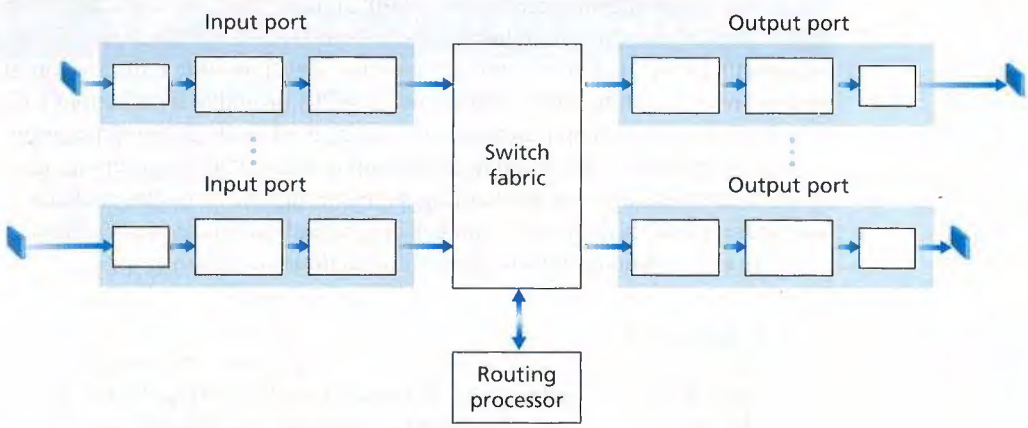


Figure 4.6 ♦ Router architecture

performs the data link–layer functions (represented by the middle boxes in the input and output ports) needed to interoperate with the data link–layer functions at the remote side of the incoming link. It also performs a lookup and forwarding function (the rightmost box of the input port and the leftmost box of the output port) so that a packet forwarded into the switching fabric of the router emerges at the appropriate output port. Control packets (for example, packets carrying routing protocol information) are forwarded from an input port to the routing processor. In practice, multiple ports are often gathered together on a single **line card** within a router.

- *Switching fabric.* The switching fabric connects the router's input ports to its output ports. This switching fabric is completely contained within the router—a network inside of a network router!
- *Output ports.* An output port stores the packets that have been forwarded to it through the switching fabric and then transmits the packets on the outgoing link. The output port thus performs the reverse data link–and physical-layer functionality of the input port. When a link is bidirectional (that is, carries traffic in both directions), an output port to the link will typically be paired with the input port for that link, on the same line card.
- *Routing processor.* The routing processor executes the routing protocols (for example, the protocols we study in Section 4.6), maintains the routing information and forwarding tables, and performs network management functions (see Chapter 9) within the router.

DELL - Exhibit 1008 - Page 108

In the following subsections, we'll look at input ports, the switching fabric, and output ports in more detail. [Chuang 2005; Keslassy 2003; Chao 2001; Turner 1988; Giacopelli 1990; McKeown 1997a; Partridge 1998] provide a discussion of some specific router architectures. [McKeown 1997b] provides a particularly readable overview of modern router architectures, using the Cisco 12000 router as an example. For concreteness, the ensuing discussion assumes that the computer network is a packet network, and that forwarding decisions are based on the packet's destination address (rather than a VC number in a virtual-circuit network). However, the concepts and techniques are similar for a virtual-circuit network.

4.3.1 Input Ports

A more detailed view of input port functionality is given in Figure 4.7. As discussed above, the input port's line termination function and data link processing implement the physical and data link layers associated with an individual input link to the router. The lookup/forwarding module in the input port is central to the forwarding function of the router. In many routers, it is here that the router determines the output port to which an arriving packet will be forwarded via the switching fabric. The choice of the output port is made using the information contained in the forwarding table. Although the forwarding table is computed by the routing processor, a shadow copy of the forwarding table is typically stored at each input port and updated, as needed, by the routing processor. With local copies of the forwarding table, the forwarding decision can be made locally, at each input port, without invoking the centralized routing processor. Such *decentralized* forwarding avoids creating a forwarding processing bottleneck at a single point within the router.

In routers with limited processing capabilities at the input port, the input port may simply forward the packet to the centralized routing processor, which will then perform the forwarding table lookup and forward the packet to the appropriate output port. This is the approach taken when a workstation or a server serves as a router; here, the routing processor is really just the workstation's CPU, and the input port is really just a network interface card (for example, an Ethernet card).

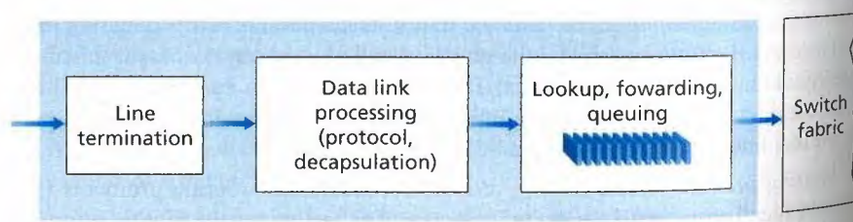


Figure 4.7 ♦ Input port processing

DELL - Exhibit 1008 - Page 109

CASE HISTORY

CISCO SYSTEMS: DOMINATING THE NETWORK CORE

As of this writing (October 2006), Cisco employs more than 30,000 people and has a market capitalization of about $140 billion. Cisco currently dominates the Internet router market and in recent years has moved into the Internet telephony market, where it competes head-to-head with the telephone equipment companies, such as Lucent, Alcatel, Nortel, and Siemens. How did this gorilla of a networking company come to be? It all started in 1984 in the living room of a Silicon Valley apartment.

Len Bosak and his wife Sandy Lerner were working at Stanford University when they had the idea to build and sell Internet routers to research and academic institutions. Sandy Lerner came up with the name Cisco (an abbreviation for San Francisco), and she also designed the company's bridge logo. Corporate headquarters was their living room, and they financed the project with credit cards and moonlighting consulting jobs. At the end of 1986, Cisco's revenues reached $250,000 a month. At the end of 1987, Cisco succeeded in attracting venture capital—$2 million from Sequoia Capital in exchange for one third of the company. Over the next few years, Cisco continued to grow and grab more and more market share. At the same time, relations between Bosak/Lerner and Cisco management became strained. Cisco went public in 1990; in the same year Lerner and Bosak left the company.

Over the years, Cisco has expanded well beyond the router market, selling security, wireless, and Voice over IP products and services. However, Cisco is facing increased international competition, including from Huawei, a rapidly growing Chinese network-gear company. Huawei, with over 38,000 employees worldwide and was recently reported to have over 7% of the Ethernet switch and router markets. Other sources of competition for Cisco in the router and switched Ethernet space includes Alcatel-Lucent, and Juniper.

Given the existence of a forwarding table, table lookup is conceptually simple—we just search through the forwarding table looking for the longest prefix match, as described in Section 4.2.2. In practice, however, life is not so simple. Perhaps the most important complicating factor is that backbone routers must operate at high speeds, performing millions of lookups per second. Indeed, it is desirable for the input port processing to be able to proceed at **line speed**, that is, for a lookup to be performed in less than the amount of time needed to receive a packet at the input port. In this case, input processing of a received packet can be completed before the next receive operation is complete. To get an idea of the performance requirements for a lookup, consider that an OC-48 link runs at 2.5 Gbps. With packets 256 bytes long, this implies a lookup speed of approximately 1 million lookups per second.

Switch fabric

DELL - Exhibit 1008 - Page 110

Given the need to operate at today's high link speeds, a linear search through a large forwarding table is impossible. A more reasonable technique is to store the forwarding table entries in a tree data structure. Each level in the tree can be thought of as corresponding to a bit in the destination address. To look up an address, one simply starts at the root node of the tree. If the first address bit is a zero, then the left subtree will contain the forwarding table entry for the destination address; otherwise it will be in the right subtree. The appropriate subtree is then traversed using the remaining address bits—if the next address bit is a zero, the left subtree of the initial subtree is chosen; otherwise, the right subtree of the initial subtree is chosen. In this manner, one can look up the forwarding table entry in N steps, where N is the number of bits in the address. (Note that this is essentially a binary search through an address space of size 2^N.) An improvement over binary search techniques is described in [Srinivasan 1999], and a general survey of packet classification algorithms can be found in [Gupta 2001].

But even with $N = 32$ (for example, a 32-bit IP address) steps, the lookup speed via binary search is not fast enough for today's backbone routing requirements. For example, assuming a memory access at each step, fewer than a million address lookups per second could be performed with 40 ns memory access times. Several techniques have thus been explored to increase lookup speeds. **Content addressable memories (CAMs)** allow a 32-bit IP address to be presented to the CAM, which returns the content of the forwarding table entry for that address in essentially constant time. The Cisco 8500 series router has a 64K CAM for each input port.

Another technique for speeding up lookup is to keep recently accessed forwarding table entries in a cache [Feldmeier 1988]. Here, the concern is the potential size of the cache. Fast data structures, which allow forwarding table entries to be located in $\log(N)$ steps [Waldvogel 1997], or which compress forwarding tables in novel ways [Brodnik 1997], have been proposed. A hardware-based approach to lookup that is optimized for the common case that the address being looked up has 24 or fewer significant bits is discussed in [Gupta 1998]. For a survey and taxonomy of high-speed IP address lookup algorithms, see [Ruiz-Sanchez 2001].

Once the output port for a packet has been determined via the lookup, the packet can be forwarded into the switching fabric. However, a packet may be temporarily blocked from entering the switching fabric (due to the fact that packets from other input ports are currently using the fabric). A blocked packet must thus be queued at the input port and then scheduled to cross the switching fabric at a later point in time. We'll take a closer look at the blocking, queuing, and scheduling of packets (at both input ports and output ports) within a router in Section 4.3.4.

4.3.2 Switching Fabric

The switching fabric is at the very heart of a router. It is through the switching fabric that the packets are actually switched (that is, forwarded) from an input port to

DELL - Exhibit 1008 - Page 111

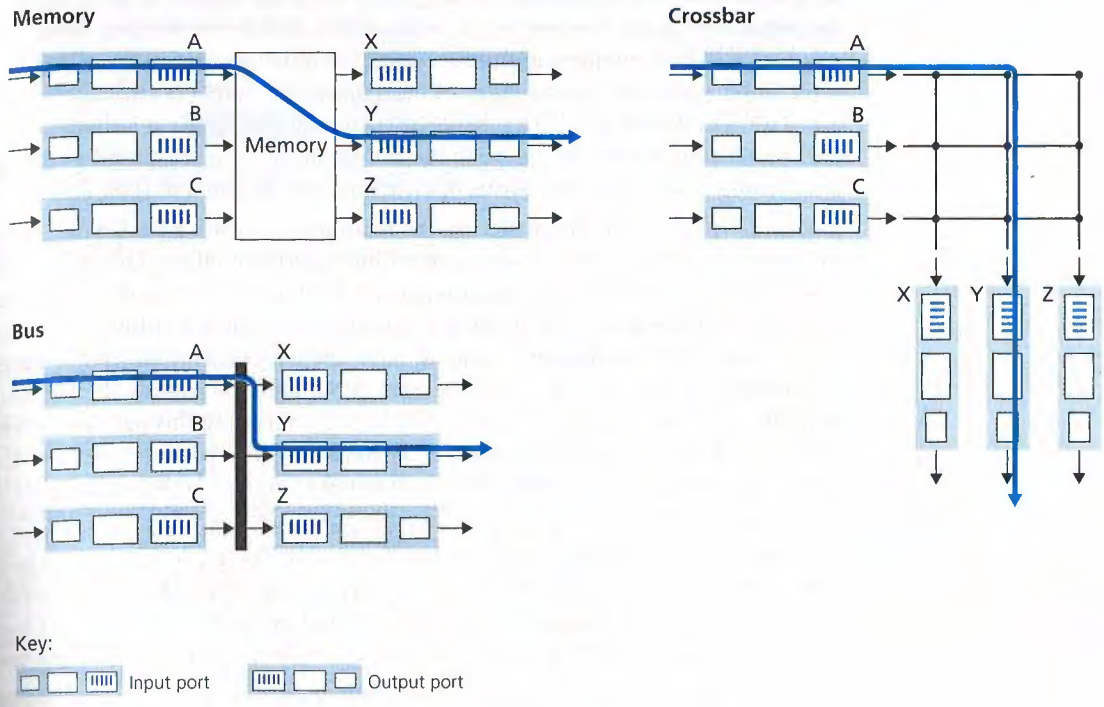


Figure 4.8 ♦ Three switching techniques

an output port. Switching can be accomplished in a number of ways, as indicated in Figure 4.8:

- *Switching via memory.* The simplest, earliest routers were often traditional computers, with switching between input and output ports being done under direct control of the CPU (routing processor). Input and output ports functioned as traditional I/O devices in a traditional operating system. An input port with an arriving packet first signaled the routing processor via an interrupt. The packet was then copied from the input port into processor memory. The routing processor then extracted the destination address from the header, looked up the appropriate output port in the forwarding table, and copied the packet to the output port's buffers. Note that if the memory bandwidth is such that B packets per second can be written into, or read from, memory, then the overall forwarding throughput (the total rate at which packets are transferred from input ports to output ports) must be less than $B/2$.

  Many modern routers also switch via memory. A major difference from early routers, however, is that the lookup of the destination address and the storing of

DELL - Exhibit 1008 - Page 112

the packet into the appropriate memory location is performed by processors on the input line cards. In some ways, routers that switch via memory look very much like shared-memory multiprocessors, with the processors on a line card switching packets into the memory of the appropriate output port. Cisco's Catalyst 8500 series switches [Cisco 8500 2007] forward packets via a shared memory. An abstract model for studying the properties of memory-based switching and a comparison with other forms of switching can be found in [Iyer 2002].

- *Switching via a bus.* In this approach, the input ports transfer a packet directly to the output port over a shared bus, without intervention by the routing processor (note that when switching via memory, the packet must also cross the system bus going to/from memory). Although the routing processor is not involved in the bus transfer, because the bus is shared, only one packet at a time can be transferred over the bus. A packet arriving at an input port and finding the bus busy with the transfer of another packet is blocked from passing through the switching fabric and is queued at the input port. Because every packet must cross the single bus, the switching bandwidth of the router is limited to the bus speed.

  Given that bus bandwidths of over 1 Gbps are possible in today's technology, switching via a bus is often sufficient for routers that operate in access and enterprise networks (for example, local area and corporate networks). Bus-based switching has been adopted in a number of current router products, including the Cisco 5600 [Cisco Switches 2007], which switches packets over a 32 Gbps backplane bus.

- *Switching via an interconnection network.* One way to overcome the bandwidth limitation of a single, shared bus is to use a more sophisticated interconnection network, such as those that have been used in the past to interconnect processors in a multiprocessor computer architecture. A crossbar switch is an interconnection network consisting of $2n$ buses that connect n input ports to n output ports, as shown in Figure 4.8. A packet arriving at an input port travels along the horizontal bus attached to the input port until it intersects with the vertical bus leading to the desired output port. If the vertical bus leading to the output port is free, the packet is transferred to the output port. If the vertical bus is being used to transfer a packet from another input port to this same output port, the arriving packet is blocked and must be queued at the input port.

  Delta and Omega switching fabrics have also been proposed as an interconnection network between input and output ports. See [Tobagi 1990] for a survey of switch architectures. Cisco 12000 family switches [Cisco 12000 2007] use an interconnection network, providing up to 60 Gbps through the switching fabric. One trend in interconnection network design [Keshav 1998] is to fragment a variable-length IP packet into fixed-length cells, then tag and switch the fixed-length cells through the interconnection network. The cells are then reassembled into the original packet at the output port. The fixed-length cell and internal tag can considerably simplify and speed up the switching of the packet through the interconnection network.

DELL - Exhibit 1008 - Page 113

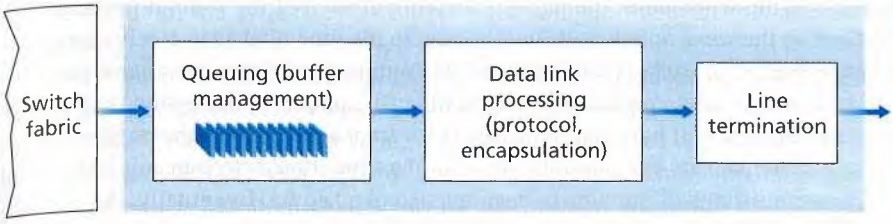


Figure 4.9 ♦ Output port processing

4.3.3 Output Ports

Output port processing, shown in Figure 4.9, takes the packets that have been stored in the output port's memory and transmits them over the outgoing link. The data link protocol processing and line termination are the send-side link- and physical-layer functionality that interacts with the input port on the other end of the outgoing link, as discussed in Section 4.3.1. The queuing and buffer management functionality is needed when the switch fabric delivers packets to the output port at a rate that exceeds the output link rate; we'll cover output port queuing below.

4.3.4 Where Does Queuing Occur?

If we look at the input and output port functionality and the configurations shown in Figure 4.8, it is evident that packet queues can form at both the input ports *and* the output ports. It is important to consider these queues in a bit more detail, since as these queues grow large, the router's buffer space will eventually be exhausted and **packet loss** will occur. Recall that in our earlier discussions, we said that packets were lost within the network or dropped at a router. It is here, at these queues within a router, where such packets are actually dropped and lost. The actual location of packet loss (either at the input port queues or the output port queues) will depend on the traffic load, the relative speed of the switching fabric, and the line speed, as discussed below.

Suppose that the input line speeds and output line speeds are all identical, and that there are n input ports and n output ports. Define the **switching fabric speed** as the rate at which the switching fabric can move packets from input ports to output ports. If the switching fabric speed is at least n times as fast as the input line speed, then no queuing can occur at the input ports. This is because even in the worst case, where all n input lines are receiving packets, the switch will be able to transfer n packets from input port to output port in the time it takes each of the n input ports to (simultaneously) receive a *single* packet. But what can happen at the output ports? Let us suppose still that the switching fabric is at least n times as fast as the line

DELL - Exhibit 1008 - Page 114

speeds. In the worst case, the packets arriving at each of the n input ports will be destined to the *same* output port. In this case, in the time it takes to receive (or send) a single packet, n packets will arrive at this output port. Since the output port can transmit only a single packet in a unit of time (the packet transmission time), the n arriving packets will have to queue (wait) for transmission over the outgoing link. Then n more packets can possibly arrive in the time it takes to transmit just one of the n packets that had previously been queued. And so on. Eventually, the number of queued packets can grow large enough to exhaust the memory space at the output port, in which case packets are dropped.

Output port queuing is illustrated in Figure 4.10. At time t, a packet has arrived at each of the incoming input ports, each destined for the uppermost outgoing port. Assuming identical line speeds and a switch operating at three times the line speed, one time unit later (that is, in the time needed to receive or send a packet), all three original packets have been transferred to the outgoing port and are queued awaiting transmission. In the next time unit, one of these three packets will have been transmitted over the outgoing link. In our example, two *new* packets have arrived at the

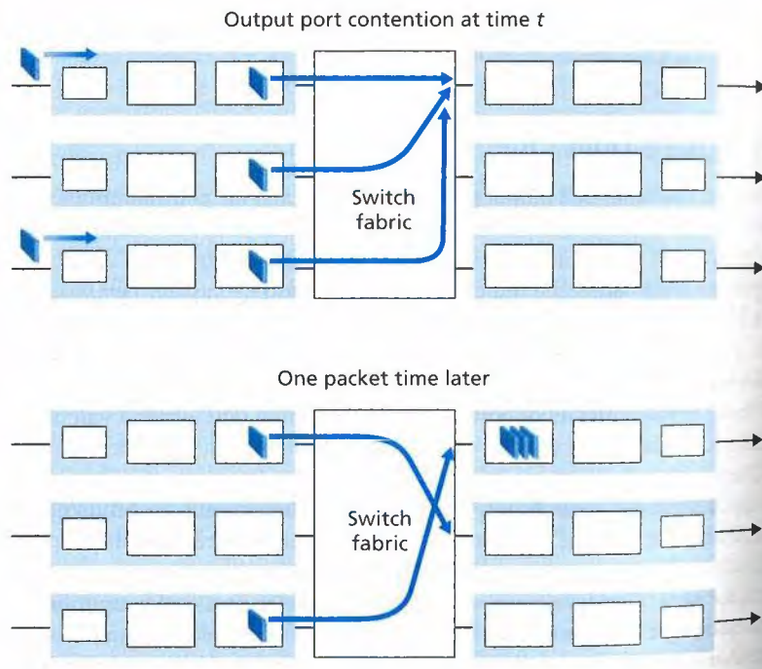


Figure 4.10 ♦ Output port queuing

DELL - Exhibit 1008 - Page 115

incoming side of the switch; one of these packets is destined for this uppermost output port.

Given that router buffers are needed to absorb the fluctuations in traffic load, the natural question to ask is how *much* buffering is required. For many years, the rule of thumb [RFC 3439] for buffer sizing was that the amount of buffering *(B)* should be equal to an average round-trip time (*RTT*, say 250 msec) times the link capacity *(C)*. This result is based on an analysis of the queueing dynamics of a relatively small number of TCP flows [Villamizar 1994]. Thus, a 10 Gbps link with an RTT of 250 msec would need an amount of buffering equal to $B = RTT \cdot C = 2.5$ Gbps of buffers. Recent theoretical and experimental efforts [Appenzeller 2004], however, suggest that when there are a large number of TCP flows (N) passing through a link, the amount of buffering needed is $B = RTT \cdot C/\sqrt{N}$. With a large number of flows typically passing through large backbone router links (see, e.g., [Fraleigh 2003]), the value of N can be large, with the decrease in needed buffer size becoming quite significant. [Appenzeller 2004] and [Wischik 2005] provide very readable discussions of the buffer sizing problem from a theoretical, implementation, and operational standpoint.

A consequence of output port queuing is that a **packet scheduler** at the output port must choose one packet among those queued for transmission. This selection might be done on a simple basis, such as first-come-first-served (FCFS) scheduling, or a more sophisticated scheduling discipline such as weighted fair queuing (WFQ), which shares the outgoing link fairly among the different end-to-end connections that have packets queued for transmission. Packet scheduling plays a crucial role in providing **quality-of-service guarantees**. We'll thus cover packet scheduling extensively in Chapter 7. A discussion of output port packet scheduling disciplines is [Cisco Queue 2007].

Similarly, if there is not enough memory to buffer an incoming packet, a decision must be made to either drop the arriving packet (a policy known as **drop-tail**) or remove one or more already-queued packets to make room for the newly arrived packet. In some cases, it may be advantageous to drop (or mark the header of) a packet *before* the buffer is full in order to provide a congestion signal to the sender. A number of packet-dropping and -marking policies (which collectively have become known as **active queue management** (**AQM**) algorithms) have been proposed and analyzed [Labrador 1999, Hollot 2002]. One of the most widely studied and implemented AQM algorithms is the **Random Early Detection** (**RED**) algorithm. Under RED, a weighted average is maintained for the length of the output queue. If the average queue length is less than a minimum threshold, min_{th}, when a packet arrives, the packet is admitted to the queue. Conversely, if the queue is full or the average queue length is greater than a maximum threshold, max_{th}, when a packet arrives, the packet is marked or dropped. Finally, if the packet arrives to find an average queue length in the interval $[min_{th}, max_{th}]$, the packet is marked or dropped with a probability that is typically some function of the average queue

DELL - Exhibit 1008 - Page 116

length, min_{th}, and max_{th}. A number of probabilistic marking/dropping functions have been proposed, and various versions of RED have been analytically modeled, simulated, and/or implemented. [Christiansen 2001] and [Floyd 2007] provide overviews and pointers to additional reading.

If the switch fabric is not fast enough (relative to the input line speeds) to transfer *all* arriving packets through the fabric without delay, then packet queuing can also occur at the input ports, as packets must join input port queues to wait their turn to be transferred through the switching fabric to the output port. To illustrate an important consequence of this queuing, consider a crossbar switching fabric and suppose that (1) all link speeds are identical, (2) that one packet can be transferred from any one input port to a given output port in the same amount of time it takes for a packet to be received on an input link, and (3) packets are moved from a given input queue to their desired output queue in an FCFS manner. Multiple packets can be transferred in parallel, as long as their output ports are different. However, if two packets at the front of two input queues are destined for the same output queue, then one of the packets will be blocked and must wait at the input queue—the switching fabric can transfer only one packet to a given output port at a time.

Figure 4.11 shows an example in which two packets (darkly shaded) at the front of their input queues are destined for the same upper-right output port. Suppose that the switch fabric chooses to transfer the packet from the front of the upper-left queue. In this case, the darkly shaded packet in the lower-left queue must wait. But not only must this darkly shaded packet wait, so too must the lightly shaded packet that is queued behind that packet in the lower-left queue, even though there is *no* contention for the middle-right output port (the destination for the lightly shaded packet). This phenomenon is known as **head-of-the-line (HOL) blocking** in an input-queued switch—a queued packet in an input queue must wait for transfer through the fabric (even though its output port is free) because it is blocked by another packet at the head of the line. [Karol 1987] shows that due to HOL blocking, the input queue will grow to unbounded length (informally, this is equivalent to saying that significant packet loss will occur) under certain assumptions as soon as the packet arrival rate on the input links reaches only 58 percent of their capacity. A number of solutions to HOL blocking are discussed in [McKeown 1997b].

4.4 The Internet Protocol (IP): Forwarding and Addressing in the Internet

Our discussion of network-layer addressing and forwarding thus far has been without reference to any specific computer network. In this section, we'll turn our attention to how addressing and forwarding are done in the Internet. We'll see that Internet addressing and forwarding are important components of the Internet

DELL - Exhibit 1008 - Page 117

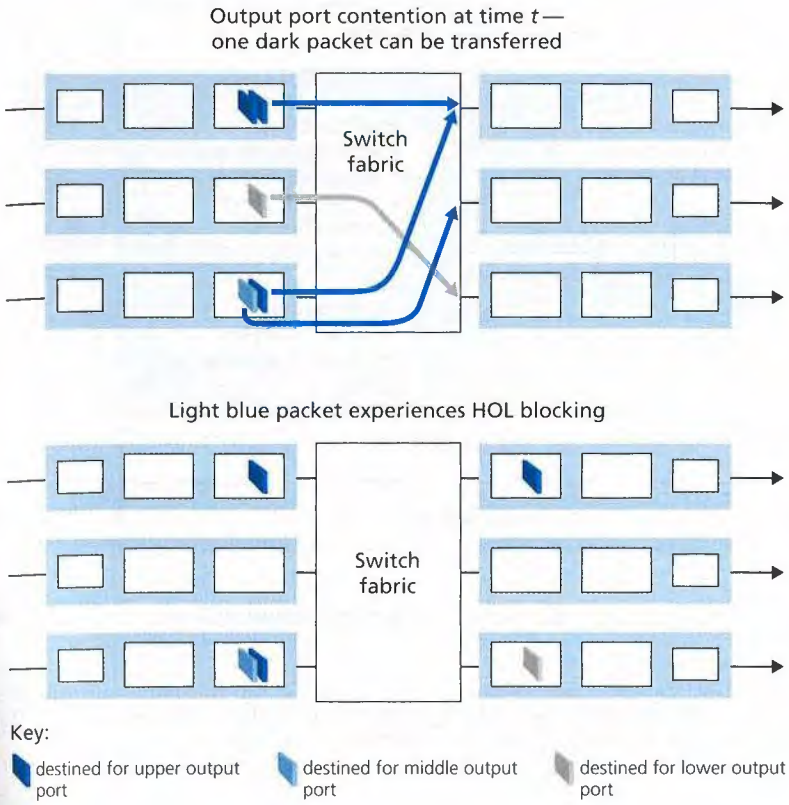


Figure 4.11 ♦ HOL blocking at an input queued switch

Protocol (IP). There are two versions of IP in use today. We'll first examine the widely deployed IP protocol version 4, which is usually referred to simply as IPv4 [RFC 791]. We'll examine IP version 6 [RFC 2460; RFC 3513], which has been proposed to replace IPv4, at the end of this section.

But before beginning our foray into IP, let's take a step back and consider the components that make up the Internet's network layer. As shown in Figure 4.12, the Internet's network layer has three major components. The first component is the IP protocol, the topic of this section. The second major component is the routing component, which determines the path a datagram follows from source to destination. We mentioned earlier that routing protocols compute the forwarding tables that are used to forward packets through the network. We'll study the Internet's routing protocols in Section 4.6. The final component of the network layer is a facility to report errors in datagrams and respond to requests for certain network-layer information.

DELL - Exhibit 1008 - Page 118

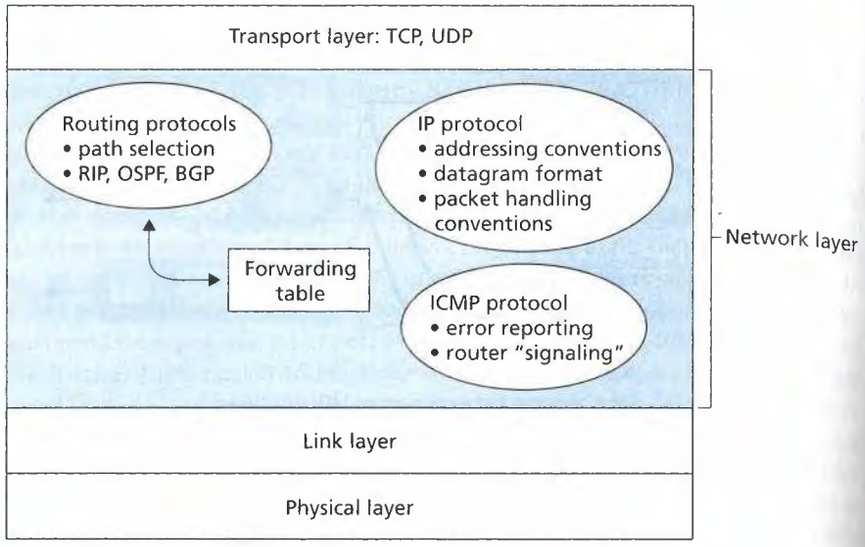


Figure 4.12 ♦ A look inside the Internet's network layer

We'll cover the Internet's network-layer error- and information-reporting protocol, the Internet Control Message Protocol (ICMP), in Section 4.4.3.

4.4.1 Datagram Format

Recall that a network-layer packet is referred to as a *datagram*. We begin our study of IP with an overview of the syntax and semantics of the IPv4 datagram. You might be thinking that nothing could be drier than the syntax and semantics of a packet's bits. Nevertheless, the datagram plays a central role in the Internet—every networking student and professional needs to see it, absorb it, and master it. The IPv4 datagram format is shown in Figure 4.13. The key fields in the IPv4 datagram are the following:

- *Version number.* These 4 bits specify the IP protocol version of the datagram. By looking at the version number, the router can determine how to interpret the remainder of the IP datagram. Different versions of IP use different datagram formats. The datagram format for the current version of IP, IPv4, is shown in Figure 4.13. The datagram format for the new version of IP (IPv6) is discussed at the end of this section.
- *Header length.* Because an IPv4 datagram can contain a variable number of options (which are included in the IPv4 datagram header), these 4 bits are needed

DELL - Exhibit 1008 - Page 119

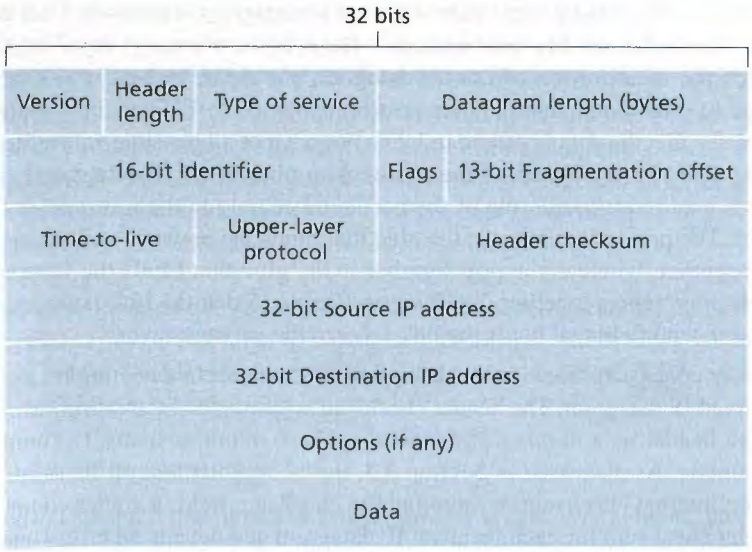


Figure 4.13 ♦ IPv4 datagram format

to determine where in the IP datagram the data actually begins. Most IP datagrams do not contain options, so the typical IP datagram has a 20-byte header.

- *Type of service.* The type of service (TOS) bits were included in the IPv4 header to allow different types of IP datagrams (for example, datagrams particularly requiring low delay, high throughput, or reliability) to be distinguished from each other. For example, it might be useful to distinguish real-time datagrams (such as those used by an IP telephony application) from non-real-time traffic (for example, FTP). The specific level of service to be provided is a policy issue determined by the router's administrator. We'll explore the topic of differentiated service in detail in Chapter 7.
- *Datagram length.* This is the total length of the IP datagram (header plus data), measured in bytes. Since this field is 16 bits long, the theoretical maximum size of the IP datagram is 65,535 bytes. However, datagrams are rarely larger than 1,500 bytes.
- *Identifier, flags, fragmentation offset.* These three fields have to do with so-called IP fragmentation, a topic we will consider in depth shortly. Interestingly, the new version of IP, IPv6, does not allow for fragmentation at routers.
- *Time-to-live.* The time-to-live (TTL) field is included to ensure that datagrams do not circulate forever (due to, for example, a long-lived routing loop) in the network. This field is decremented by one each time the datagram is processed by a router. If the TTL field reaches 0, the datagram must be dropped.

DELL - Exhibit 1008 - Page 120

- *Protocol.* This field is used only when an IP datagram reaches its final destination. The value of this field indicates the specific transport-layer protocol to which the data portion of this IP datagram should be passed. For example, a value of 6 indicates that the data portion is passed to TCP, while a value of 17 indicates that the data is passed to UDP. For a list of all possible values, see [RFC 1700; RFC 3232]. Note that the protocol number in the IP datagram has a role that is analogous to the role of the port number field in the transport-layer segment. The protocol number is the glue that binds the network and transport layers together, whereas the port number is the glue that binds the transport and application layers together. We'll see in Chapter 5 that the link-layer frame also has a special field that binds the link layer to the network layer.
- *Header checksum.* The header checksum aids a router in detecting bit errors in a received IP datagram. The header checksum is computed by treating each 2 bytes in the header as a number and summing these numbers using 1s complement arithmetic. As discussed in Section 3.3, the 1s complement of this sum, known as the Internet checksum, is stored in the checksum field. A router computes the header checksum for each received IP datagram and detects an error condition if the checksum carried in the datagram header does not equal the computed checksum. Routers typically discard datagrams for which an error has been detected. Note that the checksum must be recomputed and stored again at each router, as the TTL field, and possibly the options field as well, may change. An interesting discussion of fast algorithms for computing the Internet checksum is [RFC 1071]. A question often asked at this point is, why does TCP/IP perform error checking at both the transport and network layers? There are several reasons for this repetition. First, note that only the IP header is checksummed at the IP layer, while the TCP/UDP checksum is computed over the entire TCP/UDP segment. Second, TCP/UDP and IP do not necessarily both have to belong to the same protocol stack. TCP can, in principle, run over a different protocol (for example, ATM) and IP can carry data that will not be passed to TCP/UDP.
- *Source and destination IP addresses.* When a source creates a datagram, it inserts its IP address into the source IP address field and inserts the address of the ultimate destination into the destination IP address field. Often the source host determines the destination address via a DNS lookup, as discussed in Chapter 2. We'll discuss IP addressing in detail in Section 4.4.2.
- *Options.* The options fields allow an IP header to be extended. Header options were meant to be used rarely—hence the decision to save overhead by not including the information in options fields in every datagram header. However, the mere existence of options does complicate matters—since datagram headers can be of variable length, one cannot determine a priori where the data field will start. Also, since some datagrams may require options processing and others may not, the amount of time needed to process an IP datagram at a router can vary

DELL - Exhibit 1008 - Page 121

greatly. These considerations become particularly important for IP processing in high-performance routers and hosts. For these reasons and others, IP options were dropped in the IPv6 header, as discussed in Section 4.4.4.

- *Data (payload).* Finally, we come to the last and most important field—the *raison d'être* for the datagram in the first place! In most circumstances, the data field of the IP datagram contains the transport-layer segment (TCP or UDP) to be delivered to the destination. However, the data field can carry other types of data, such as ICMP messages (discussed in Section 4.4.3).

Note that an IP datagram has a total of 20 bytes of header (assuming no options). If the datagram carries a TCP segment, then each (nonfragmented) datagram carries a total of 40 bytes of header (20 bytes of IP header plus 20 bytes of TCP header) along with the application-layer message.

IP Datagram Fragmentation

We'll see in Chapter 5 that not all link-layer protocols can carry network-layer packets of the same size. Some protocols can carry big datagrams, whereas other protocols can carry only little packets. For example, Ethernet frames can carry up to 1,500 bytes of data, whereas frames for some wide-area links can carry no more than 576 bytes. The maximum amount of data that a link-layer frame can carry is called the maximum transmission unit (MTU). Because each IP datagram is encapsulated within the link-layer frame for transport from one router to the next router, the MTU of the link-layer protocol places a hard limit on the length of an IP datagram. Having a hard limit on the size of an IP datagram is not much of a problem. What is a problem is that each of the links along the route between sender and destination can use different link-layer protocols, and each of these protocols can have different MTUs.

To understand the forwarding issue better, imagine that *you* are a router that interconnects several links, each running different link-layer protocols with different MTUs. Suppose you receive an IP datagram from one link. You check your forwarding table to determine the outgoing link, and this outgoing link has an MTU that is smaller than the length of the IP datagram. Time to panic—how are you going to squeeze this oversized IP datagram into the payload field of the link-layer frame? The solution is to fragment the data in the IP datagram into two or more smaller IP datagrams, encapsulate each of these smaller IP datagrams in a separate link-layer frame; and send these frames over the outgoing link. Each of these smaller datagrams is referred to as a **fragment**.

Fragments need to be reassembled before they reach the transport layer at the destination. Indeed, both TCP and UDP are expecting to receive complete, unfragmented segments from the network layer. The designers of IPv4 felt that reassembling datagrams in the routers would introduce significant complication into the

DELL - Exhibit 1008 - Page 122

protocol and put a damper on router performance. (If you were a router, would you want to be reassembling fragments on top of everything else you had to do?) Sticking to the principle of keeping the network core simple, the designers of IPv4 decided to put the job of datagram reassembly in the end systems rather than in network routers.

When a destination host receives a series of datagrams from the same source, it needs to determine whether any of these datagrams are fragments of some original, larger datagram. If some datagrams are fragments, it must further determine when it has received the last fragment and how the fragments it has received should be pieced back together to form the original datagram. To allow the destination host to perform these reassembly tasks, the designers of IP (version 4) put *identification, flag,* and *fragmentation offset* fields in the IP datagram header. When a datagram is created, the sending host stamps the datagram with an identification number as well as source and destination addresses. Typically, the sending host increments the identification number for each datagram it sends. When a router needs to fragment a datagram, each resulting datagram (that is, fragment) is stamped with the source address, destination address, and identification number of the original datagram. When the destination receives a series of datagrams from the same sending host, it can examine the identification numbers of the datagrams to determine which of the datagrams are actually fragments of the same larger datagram. Because IP is an unreliable service, one or more of the fragments may never arrive at the destination. For this reason, in order for the destination host to be absolutely sure it has received the last fragment of the original datagram, the last fragment has a flag bit set to 0, whereas all the other fragments have this flag bit set to 1. Also, in order for the destination host to determine whether a fragment is missing (and also to be able to reassemble the fragments in their proper order), the offset field is used to specify where the fragment fits within the original IP datagram.

Figure 4.14 illustrates an example. A datagram of 4,000 bytes (20 bytes of IP header plus 3,980 bytes of IP payload) arrives at a router and must be forwarded to a link with an MTU of 1,500 bytes. This implies that the 3,980 data bytes in the original datagram must be allocated to three separate fragments (each of which is also an IP datagram). Suppose that the original datagram is stamped with an identification number of 777. The characteristics of the three fragments are shown in Table 4.2. The values in Table 4.2 reflect the requirement that the amount of original payload data in all but the last fragment be a multiple of 8 bytes, and that the offset value be specified in units of 8-byte chunks.

At the destination, the payload of the datagram is passed to the transport layer only after the IP layer has fully reconstructed the original IP datagram. If one or more of the fragments does not arrive at the destination, the incomplete datagram is discarded and not passed to the transport layer. But, as we learned in the previous chapter, if TCP is being used at the transport layer, then TCP will recover from this loss by having the source retransmit the data in the original datagram.

DELL - Exhibit 1008 - Page 123

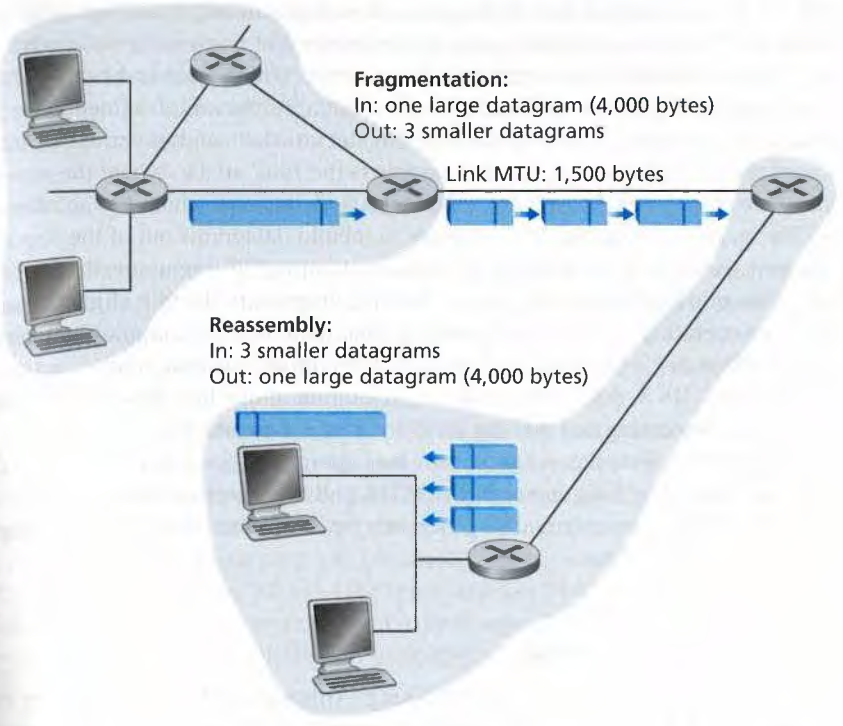


Figure 4.14 ♦ IP fragmentation and reassembly

Fragment	Bytes	ID	Offset	Flag
1st fragment	1,480 bytes in the data field of the IP datagram	identification = 777	offset = 0 (meaning the data should be inserted beginning at byte 0)	flag = 1 (meaning there is more)
2nd fragment	1,480 bytes of data	identification = 777	offset = 185 (meaning the data should be inserted beginning at byte 1,480. Note that 185 · 8 = 1,480)	flag = 1 (meaning there is more)
3rd fragment	1,020 bytes (= 3,980–1,480–1,480) of data	identification = 777	offset = 370 (meaning the data should be inserted beginning at byte 2,960. Note that 370 · 8 = 2,960)	flag = 0 (meaning this is the last fragment)

Table 4.2 ♦ IP fragments

DELL - Exhibit 1008 - Page 124

We have just learned that IP fragmentation plays an important role in gluing together the many disparate link-layer technologies. But fragmentation also has its costs. First, it complicates routers and end systems, which need to be designed to accommodate datagram fragmentation and reassembly. Second, fragmentation can be used to create lethal DoS attacks, whereby the attacker sends a series of bizarre and unexpected fragments. A classic example is the Jolt2 attack, where the attacker sends a stream of small fragments to the target host, none of which has an offset of zero. The target can collapse as it attempts to rebuild datagrams out of the degenerate packets. Another class of exploits sends overlapping IP fragments, that is, fragments whose offset values are set so that the fragments do not align properly. Vulnerable operating systems, not knowing what to do with overlapping fragments, can crash [Skoudis 2006]. As we'll see at the end of this section, a new version of the IP protocol, IPv6, does away with fragmentation altogether, thereby streamlining IP packet processing and making IP less vulnerable to attack.

At this book's Web site, we provide a Java applet that generates fragments. You provide the incoming datagram size, the MTU, and the incoming datagram identification. The applet automatically generates the fragments for you. See http://www.awl.com/kurose-ross.

4.4.2 IPv4 Addressing

We now turn our attention to IPv4 addressing. Although you may be thinking that addressing must be a straightforward topic, hopefully by the end of this chapter you'll be convinced that Internet addressing is not only a juicy, subtle, and interesting topic but also one that is of central importance to the Internet. Excellent treatments of IPv4 addressing are [3Com Addressing 2007] and the first chapter in [Stewart 1999].

Before discussing IP addressing, however, we'll need to say a few words about how hosts and routers are connected into the network. A host typically has only a single link into the network; when IP in the host wants to send a datagram, it does so over this link. The boundary between the host and the physical link is called an **interface**. Now consider a router and its interfaces. Because a router's job is to receive a datagram on one link and forward the datagram on some other link, a router necessarily has two or more links to which it is connected. The boundary between the router and any one of its links is also called an interface. A router thus has multiple interfaces, one for each of its links. Because every host and router is capable of sending and receiving IP datagrams, IP requires each host and router interface to have its own IP address. Thus, an IP address is technically associated with an interface, rather than with the host or router containing that interface.

Each IP address is 32 bits long (equivalently, 4 bytes), and there are thus a total of 2^{32} possible IP addresses. By approximating 2^{10} by 10^3, it is easy to see that there are about 4 billion possible IP addresses. These addresses are typically written in

DELL - Exhibit 1008 - Page 125

so-called **dotted-decimal notation**, in which each byte of the address is written in its decimal form and is separated by a period (dot) from other bytes in the address. For example, consider the IP address 193.32.216.9. The 193 is the decimal equivalent of the first 8 bits of the address; the 32 is the decimal equivalent of the second 8 bits of the address, and so on. Thus, the address 193.32.216.9 in binary notation is

11000001 00100000 11011000 00001001

Each interface on every host and router in the global Internet must have an IP address that is globally unique (except for interfaces behind NATs, as discussed at the end of this section). These addresses cannot be chosen in a willy-nilly manner, however. A portion of an interface's IP address will be determined by the subnet to which it is connected.

Figure 4.15 provides an example of IP addressing and interfaces. In this figure, one router (with three interfaces) is used to interconnect seven hosts. Take a close look at the IP addresses assigned to the host and router interfaces; there are several things to notice. The three hosts in the upper-left portion of Figure 4.15, and the router interface to which they are connected, all have an IP address of the form 223.1.1.xxx. That is, they all have the same leftmost 24 bits in their IP address. The four interfaces are also interconnected to each other by a network *that contains no routers*. (This network could be, for example, an Ethernet LAN, in which case the

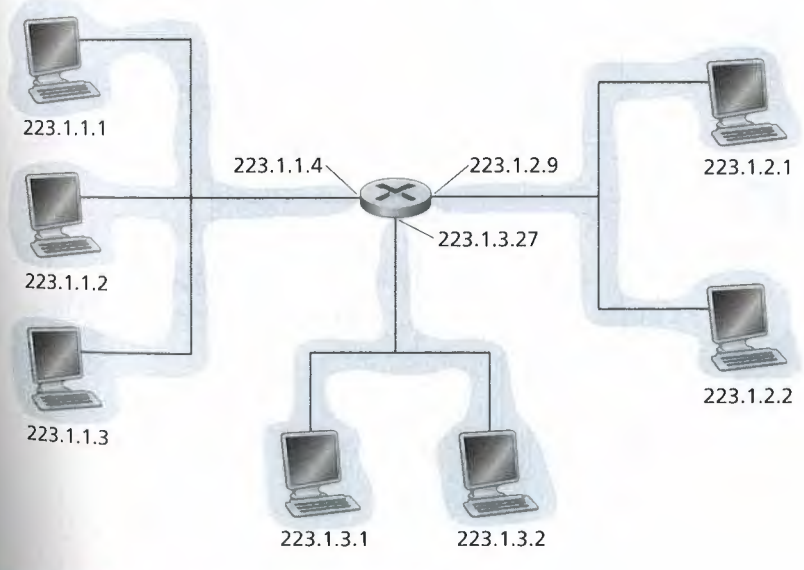


Figure 4.15 ♦ Interface addresses and subnets

DELL - Exhibit 1008 - Page 126

interfaces would be interconnected by an Ethernet hub or an Ethernet switch; see Chapter 5.) In IP terms, this network interconnecting three host interfaces and one router interface forms a **subnet** [RFC 950]. (A subnet is also called an *IP network* or simply a *network* in the Internet literature.) IP addressing assigns an address to this subnet: 223.1.1.0/24, where the /24 notation, sometimes known as a **subnet mask**, indicates that the leftmost 24 bits of the 32-bit quantity define the subnet address. The subnet 223.1.1.0/24 thus consists of the three host interfaces (223.1.1.1, 223.1.1.2, and 223.1.1.3) and one router interface (223.1.1.4). Any additional hosts attached to the 223.1.1.0/24 subnet would be *required* to have an address of the form 223.1.1.xxx. There are two additional subnets shown in Figure 4.15: the 223.1.2.0/24 network and the 223.1.3.0/24 subnet. Figure 4.16 illustrates the three IP subnets present in Figure 4.15.

The IP definition of a subnet is not restricted to Ethernet segments that connect multiple hosts to a router interface. To get some insight here, consider Figure 4.17, which shows three routers that are interconnected with each other by point-to-point links. Each router has three interfaces, one for each point-to-point link and one for the broadcast link that directly connects the router to a pair of hosts. What subnets are present here? Three subnets, 223.1.1.0/24, 223.1.2.0/24, and 223.1.3.0/24, are similar to the subnets we encountered in Figure 4.15. But note that there are three additional subnets in this example as well: one subnet, 223.1.9.0/24, for the interfaces that connect routers R1 and R2; another subnet, 223.1.8.0/24, for the interfaces that connect routers R2 and R3; and a third subnet, 223.1.7.0/24, for the interfaces that

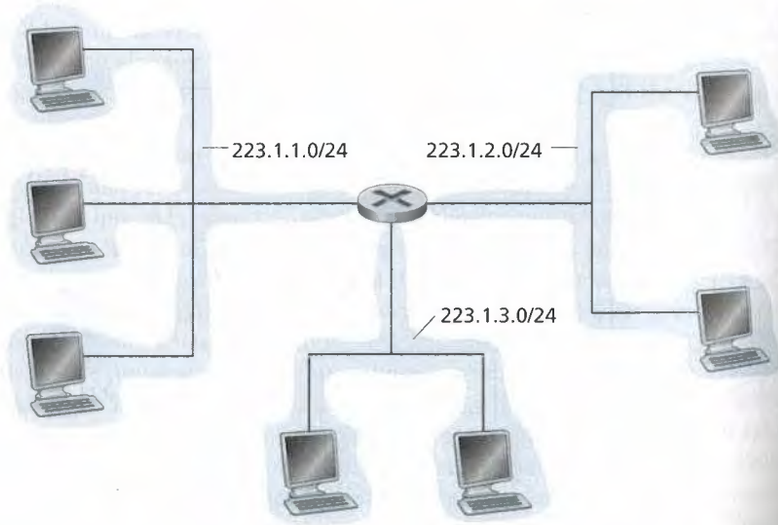


Figure 4.16 ♦ Subnet addresses

DELL - Exhibit 1008 - Page 127

connect routers R3 and R1. For a general interconnected system of routers and hosts, we can use the following recipe to define the subnets in the system:

> *To determine the subnets, detach each interface from its host or router, creating islands of isolated networks, with interfaces terminating the end points of the isolated networks. Each of these isolated networks is called a* ***subnet****.*

If we apply this procedure to the interconnected system in Figure 4.17, we get six islands or subnets.

From the discussion above, it's clear that an organization (such as a company or academic institution) with multiple Ethernet segments and point-to-point links will have multiple subnets, with all of the devices on a given subnet having the same subnet address. In principle, the different subnets could have quite different subnet addresses. In practice, however, their subnet addresses often have much in common. To understand why, let's next turn our attention to how addressing is handled in the global Internet.

The Internet's address assignment strategy is known as **Classless Interdomain Routing** (**CIDR**—pronounced *cider*) [RFC 4632]. CIDR generalizes the notion of

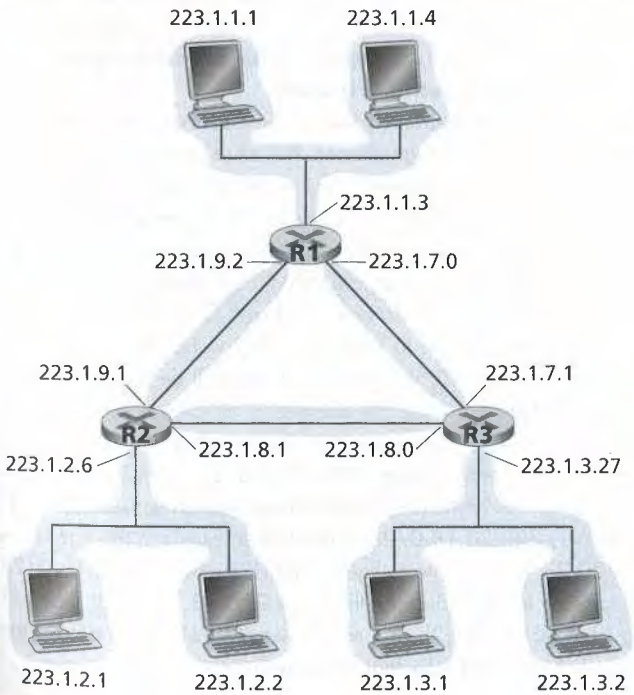


Figure 4.17 ♦ Three routers interconnecting six subnets

DELL - Exhibit 1008 - Page 128

subnet addressing. As with subnet addressing, the 32-bit IP address is divided into two parts and again has the dotted-decimal form *a.b.c.d/x*, where *x* indicates the number of bits in the first part of the address.

The *x* most significant bits of an address of the form *a.b.c.d/x* constitute the network portion of the IP address, and are often referred to as the **prefix** (or *network prefix*) of the address. An organization is typically assigned a block of contiguous addresses, that is, a range of addresses with a common prefix (see the Principles in Practice sidebar). In this case, the IP addresses of devices within the organization will share the common prefix. When we cover the Internet's BGP

PRINCIPLES IN PRACTICE

This example of an ISP that connects eight organizations to the Internet nicely illustrates how carefully allocated CIDRized addresses facilitate routing. Suppose, as shown in Figure 4.18, that the ISP (which we'll call Fly-By-Night-ISP) advertises to the outside world that it should be sent any datagrams whose first 20 address bits match 200.23.16.0/20. The rest of the world need not know that within the address block 200.23.16.0/20 there are in fact eight other organizations, each with its own subnets. This ability to use a single prefix to advertise multiple networks is often referred to as **address aggregation** (also **route aggregation** or **route summarization**).

Address aggregation works extremely well when addresses are allocated in blocks to ISPs and then from ISPs to client organizations. But what happens when addresses are not allocated in such a hierarchical manner? What would happen, for example, if Fly-By-Night-ISP acquires ISPs-R-Us and then has Organization 1 connect to the Internet through its subsidiary ISPs-R-Us? As shown in Figure 4.18, the subsidiary ISPs-R-Us owns the address block 199.31.16.0/16, but Organization 1's IP addresses are unfortunately outside of this address block. What should be done here? Certainly, Organization 1 could renumber all of its routers and hosts to have addresses within the ISPs-R-Us address block. But this is a costly solution, and Organization 1 might well be reassigned to another subsidiary in the future. The solution typically adopted is for Organization 1 to keep its IP addresses in 200.23.18.0/23. In this case, as shown in Figure 4.19, Fly-By-Night-ISP continues to advertise the address block 200.23.16.0/20 and ISPs-R-Us continues to advertise 199.31.16.0/16. However, ISPs-R-Us now *also* advertises the block of addresses for Organization 1, 200.23.18.0/23. When other routers in the larger Internet see the address blocks 200.23.16.0/20 (from Fly-By-Night-ISP) and 200.23.18.0/23 (from ISPs-R-Us) and want to route to an address in the block 200.23.18.0/23, they will use longest prefix matching (see Section 4.2.2), and route toward ISPs-R-Us, as it advertises the longest (most specific) address prefix that matches the destination address.

DELL - Exhibit 1008 - Page 129

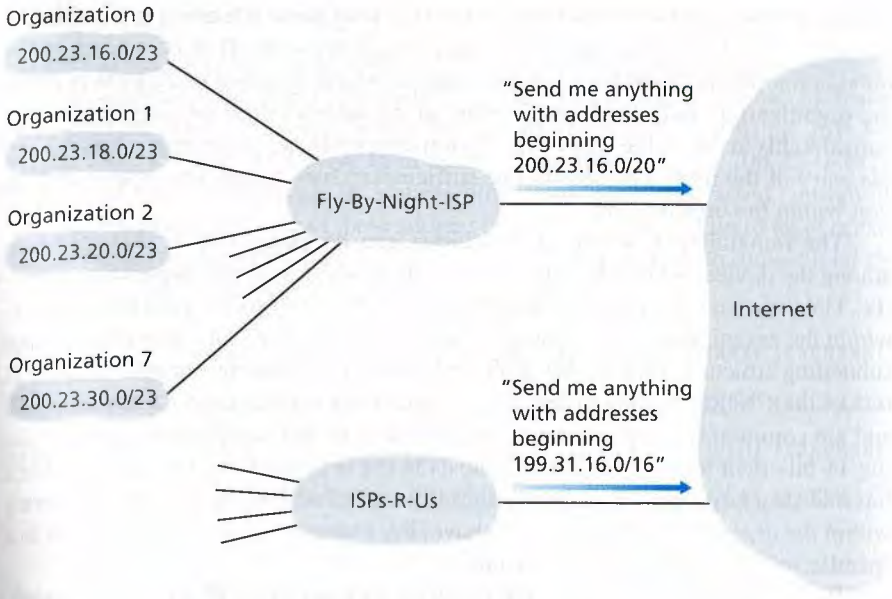


Figure 4.18 ♦ Hierarchical addressing and route aggregation

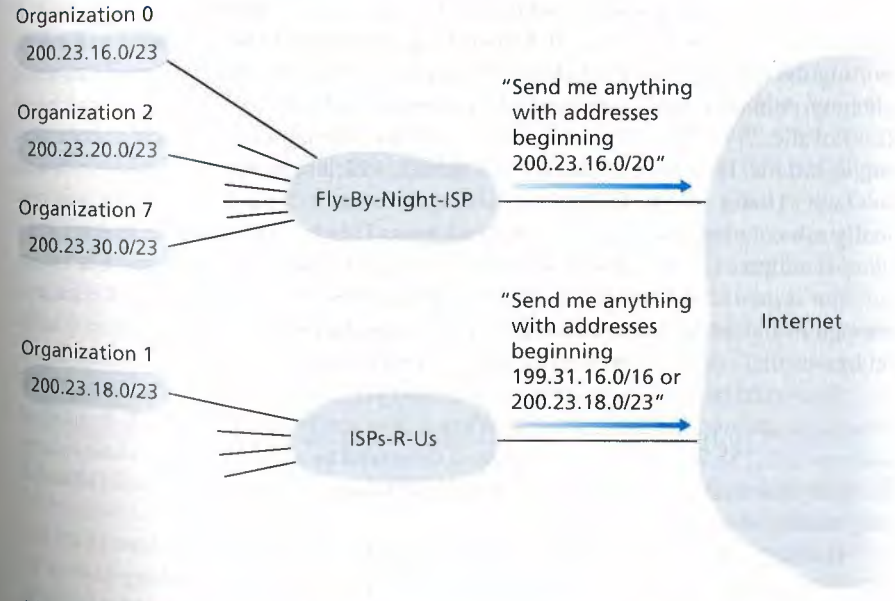


Figure 4.19 ♦ ISPs-R-Us has a more specific route to Organization 1.

DELL - Exhibit 1008 - Page 130

routing protocol in Section 4.6, we'll see that only these x leading prefix bits are considered by routers outside the organization's network. That is, when a router outside the organization forwards a datagram whose destination address is inside the organization, only the leading x bits of the address need be considered. This considerably reduces the size of the forwarding table in these routers, since a *single* entry of the form *a.b.c.d/x* will be sufficient to forward packets to *any* destination within the organization.

The remaining 32-x bits of an address can be thought of as distinguishing among the devices *within* the organization, all of which have the same network prefix. These are the bits that will be considered when forwarding packets at routers *within* the organization. These lower-order bits may (or may not) have an additional subnetting structure, such as that discussed above. For example, suppose the first 21 bits of the CIDRized address a.b.c.d/21 specify the organization's network prefix and are common to the IP addresses of all devices in that organization. The remaining 11 bits then identify the specific hosts in the organization. The organization's internal structure might be such that these 11 rightmost bits are used for subnetting within the organization, as discussed above. For example, a.b.c.d/24 might refer to a specific subnet within the organization.

Before CIDR was adopted, the network portions of an IP address were constrained to be 8, 16, or 24 bits in length, an addressing scheme known as **classful addressing**, since subnets with 8-, 16-, and 24-bit subnet addresses were known as class A, B, and C networks, respectively. The requirement that the subnet portion of an IP address be exactly 1, 2, or 3 bytes long turned out to be problematic for supporting the rapidly growing number of organizations with small and medium-sized subnets. A class C (/24) subnet could accommodate only up to $2^8 - 2 = 254$ hosts (two of the $2^8 = 256$ addresses are reserved for special use)—too small for many organizations. However, a class B (/16) subnet, which supports up 65,634 hosts, was too large. Under classful addressing, an organization with, say, 2,000 hosts was typically allocated a class B (/16) subnet address. This led to a rapid depletion of the class B address space and poor utilization of the assigned address space. For example, the organization that used a class B address for its 2,000 hosts was allocated enough of the address space for up to 65,534 interfaces—leaving more than 63,000 addresses that could not be used by other organizations.

We would be remiss if we did not mention yet another type of IP address, the IP broadcast address 255.255.255.255. When a host sends a datagram with destination address 255.255.255.255, the message is delivered to all hosts on the same subnet. Routers optionally forward the message into neighboring subnets as well (although they usually don't).

Having now studied IP addressing in detail, we need to know how hosts and subnets get their addresses in the first place. Let's begin by looking at how an organization gets a block of addresses for its devices, and then look at how a device (such as a host) is assigned an address from within the organization's block of addresses.

DELL - Exhibit 1008 - Page 131

Obtaining a Block of Addresses

In order to obtain a block of IP addresses for use within an organization's subnet, a network administrator might first contact its ISP, which would provide addresses from a larger block of addresses that had already been allocated to the ISP. For example, the ISP may itself have been allocated the address block 200.23.16.0/20. The ISP, in turn, could divide its address block into eight equal-sized contiguous address blocks and give one of these address blocks out to each of up to eight organizations that are supported by this ISP, as shown below. (We have underlined the subnet part of these addresses for your convenience.)

ISP's block	200.23.16.0/20	11001000 00010111 0001**0000** 00000000
Organization 0	200.23.16.0/23	11001000 00010111 0001000**0** 00000000
Organization 1	200.23.18.0/23	11001000 00010111 0001001**0** 00000000
Organization 2	200.23.20.0/23	11001000 00010111 0001010**0** 00000000
... ...		...
Organization 7	200.23.30.0/23	11001000 00010111 0001111**0** 00000000

While obtaining a set of addresses from an ISP is one way to get a block of addresses, it is not the only way. Clearly, there must also be a way for the ISP itself to get a block of addresses. Is there a global authority that has ultimate responsibility for managing the IP address space and allocating address blocks to ISPs and other organizations? Indeed there is! IP addresses are managed under the authority of the Internet Corporation for Assigned Names and Numbers (ICANN) [ICANN 2007], based on guidelines set forth in [RFC 2050]. The role of the nonprofit ICANN organization [NTIA 1998] is not only to allocate IP addresses, but also to manage the DNS root servers. It also has the very contentious job of assigning domain names and resolving domain name disputes. The ICANN allocates addresses to regional Internet registries (for example, ARIN, RIPE, APNIC, and LACNIC, which together form the Address Supporting Organization of ICANN [ASO-ICANN 2007]), and handle the allocation/management of addresses within their regions.

Obtaining a Host Address: the Dynamic Host Configuration Protocol

Once an organization has obtained a block of addresses, it can assign individual IP addresses to the host and router interfaces in its organization. A system administrator will typically manually configure the IP addresses into the router (often remotely, with a network management tool). Host addresses can also be configured manually, but more often this task is now done using the **Dynamic Host Configuration Protocol (DHCP)** [RFC 2131]. DHCP allows a host to obtain (be allocated) an IP address automatically. A network administrator can configure DHCP so that a

DELL - Exhibit 1008 - Page 132

given host receives the same IP address each time it connects to the network, or a host may be assigned a **temporary IP address** that will be different each time the host connects to the network. In addition to host IP address assignment, DHCP also allows a host to learn additional information, such as its subnet mask, the address of its first-hop router (often called the default gateway), and the address of its local DNS server.

Because of DHCP's ability to automate the network-related aspects of connecting a host into a network, it is often referred to as a **plug-and-play protocol**. This capability makes it *very* attractive to the network administrator who would otherwise have to perform these tasks manually! DHCP is also enjoying widespread use in residential Internet access networks and in wireless LANs, where hosts join and leave the network frequently. Consider, for example, the student who carries a laptop from a dormitory room to a library to a classroom. It is likely that in each location, the student will be connecting into a new subnet and hence will need a new IP address at each location. DHCP is ideally suited to this situation, as there are many users coming and going, and addresses are needed for only a limited amount of time. DHCP is similarly useful in residential ISP access networks. Consider, for example, a residential ISP that has 2,000 customers, but no more than 400 customers are ever online at the same time. In this case, rather than needing a block of 2,048 addresses, a DHCP server that assigns addresses dynamically needs only a block of 512 addresses (for example, a block of the form a.b.c.d/23). As the hosts join and leave, the DHCP server needs to update its list of available IP addresses. Each time a host joins, the DHCP server allocates an arbitrary address from its current pool of available addresses; each time a host leaves, its address is returned to the pool.

DHCP is a client-server protocol. A client is typically a newly arriving host wanting to obtain network configuration information, including an IP address for itself. In the simplest case, each subnet (in the addressing sense of Figure 4.17) will have a DHCP server. If no server is present on the subnet, a DHCP relay agent (typically a router) that knows the address of a DHCP server for that network is needed. Figure 4.20 shows a DHCP server attached to subnet 223.1.2/24, with the router serving as the relay agent for arriving clients attached to subnets 223.1.1/24 and 223.1.3/24. In our discussion below, we'll assume that a DHCP server is available on the subnet.

For a newly arriving host, the DHCP protocol is a four-step process, as shown in Figure 4.21 for the network setting shown in Figure 4.20. In this figure, `yiaddr` (as in "your Internet address") indicates the address being allocated to the newly arriving client. The four steps are:

- *DHCP server discovery.* The first task of a newly arriving host is to find a DHCP server with which to interact. This is done using a **DHCP discover message**, which a client sends within a UDP packet to port 67. The UDP packet is encapsulated in an IP datagram. But to whom should this datagram be sent? The host doesn't even know the IP address of the network to which it is attaching, much

DELL - Exhibit 1008 - Page 133

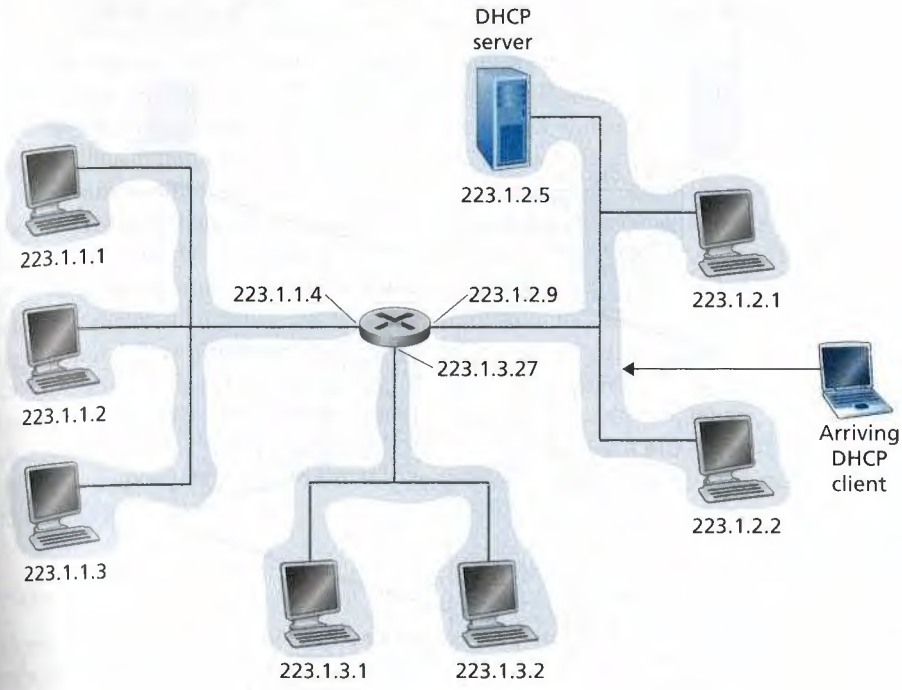


Figure 4.20 ♦ DHCP client-server scenario

less the address of a DHCP server for this network. Given this, the DHCP client creates an IP datagram containing its DHCP discover message along with the broadcast destination IP address of 255.255.255.255 and a "this host" source IP address of 0.0.0.0. The DHCP client passes the IP datagram to the link layer, which then broadcasts this frame to all nodes attached to the subnet (we will cover the details of link-layer broadcasting in Section 5.4).

- *DHCP server offer(s).* A DHCP server receiving a DHCP discover message responds to the client with a **DHCP offer message** that is broadcast to all nodes on the subnet, again using the IP broadcast address of 255.255.255.255. (You might want to think about why this server reply must also be broadcast). Since several DHCP servers can be present on the subnet, the client may find itself in the enviable position of being able to choose from among several offers. Each server offer message contains the transaction ID of the received discover message, the proposed IP address for the client, the network mask, and an IP **address lease time**—the amount of time for which the IP address will be valid. It is common for the server to set the lease time to several hours or days [Droms 1999].

DELL - Exhibit 1008 - Page 134

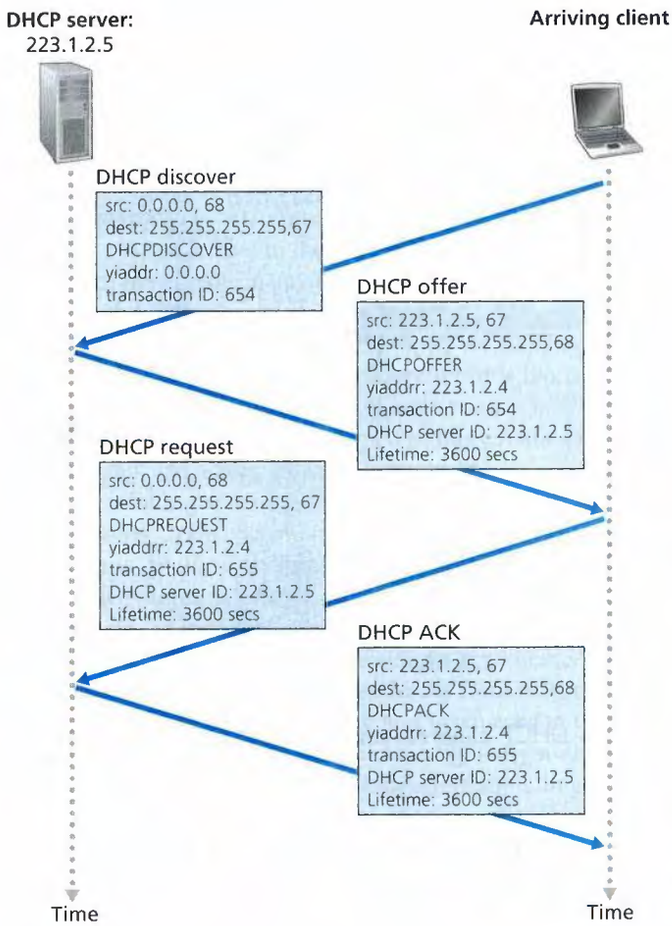


Figure 4.21 ♦ DHCP client-server interaction

- *DHCP request.* The newly arriving client will choose from among one or more server offers and respond to its selected offer with a **DHCP request message**, echoing back the configuration parameters.
- *DHCP ACK.* The server responds to the DHCP request message with a **DHCP ACK message**, confirming the requested parameters.

Once the client receives the DHCP ACK, the interaction is complete and the client can use the DHCP-allocated IP address for the lease duration. Since a client

DELL - Exhibit 1008 - Page 135

may want to use its address beyond the lease's expiration, DHCP also provides a mechanism that allows a client to renew its lease on an IP address.

The value of DHCP's plug-and-play capability is clear, considering the fact that the alternative is to manually configure a host's IP address. Consider the student who moves from classroom to library to dorm room with a laptop, joins a new subnet, and thus obtains a new IP address at each location. It is unimaginable that a system administrator would have to reconfigure laptops at each location, and few students (except those taking a computer networking class!) would have the expertise to configure their laptops manually. From a mobility aspect, however, DHCP does have shortcomings. Since a new IP address is obtained from DHCP each time a node connects to a new subnet, a TCP connection to a remote application cannot be maintained as a mobile node moves between subnets. In Chapter 6, we will examine mobile IP—a recent extension to the IP infrastructure that allows a mobile node to use a single permanent address as it moves between subnets. Additional details about DHCP can be found in [Droms 1999] and [dhc 2007]. An open source reference implementation of DHCP is available from the Internet Systems Consortium [ISC 2007].

Network Address Translation (NAT)

Given our discussion about Internet addresses and the IPv4 datagram format, we're now well aware that every IP-capable device needs an IP address. With the proliferation of small office, home office (SOHO) subnets, this would seem to imply that whenever a SOHO wants to install a LAN to connect multiple machines, a range of addresses would need to be allocated by the ISP to cover all of the SOHO's machines. If the subnet grew bigger (for example, the kids at home have not only their own computers, but have bought handheld PDAs, IP-capable phones, and networked Game Boys as well), a larger block of addresses would have to be allocated. But what if the ISP had already allocated the contiguous portions of the SOHO network's current address range? And what typical homeowner wants (or should need) to know how to manage IP addresses in the first place? Fortunately, there is a simpler approach to address allocation that has found increasingly widespread use in such scenarios: **network address translation** (**NAT**) [RFC 2663; RFC 3022].

Figure 4.22 shows the operation of a NAT-enabled router. The NAT-enabled router, residing in the home, has an interface that is part of the home network on the right of Figure 4.22. Addressing within the home network is exactly as we have seen above—all four interfaces in the home network have the same subnet address of 10.0.0/24. The address space 10.0.0.0/8 is one of three portions of the IP address space that is reserved in [RFC 1918] for a private network or a **realm** with private addresses, such as the home network in Figure 4.22. A *realm with private addresses* refers to a network whose addresses only have meaning to devices within that network. To see why this is important, consider the fact that there are hundreds of

DELL - Exhibit 1008 - Page 136

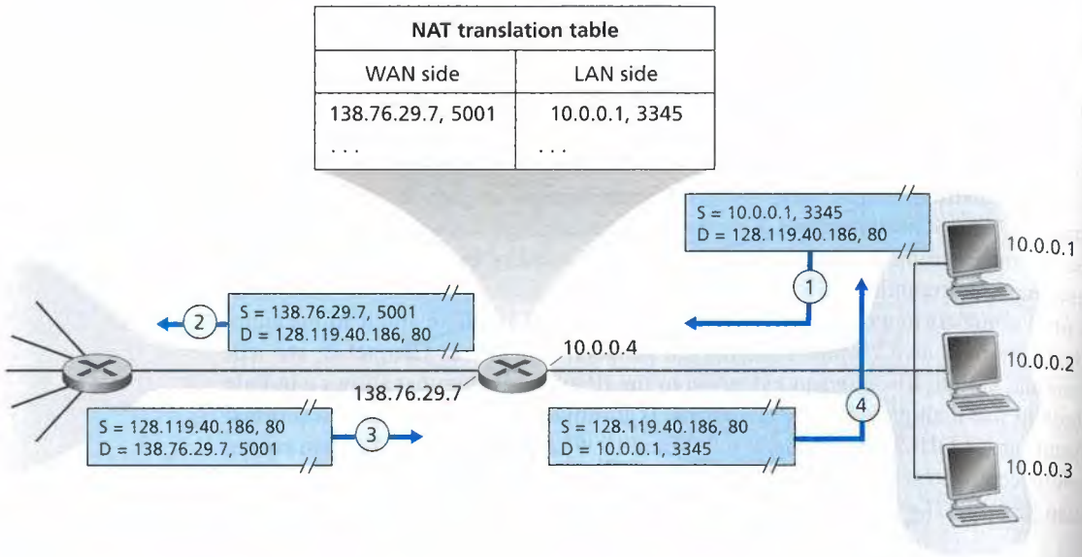


Figure 4.22 ♦ Network address translation

thousands of home networks, many using the same address space, 10.0.0.0/24. Devices within a given home network can send packets to each other using 10.0.0.0/24 addressing. However, packets forwarded *beyond* the home network into the larger global Internet clearly cannot use these addresses (as either a source or a destination address) because there are hundreds of thousands of networks using this block of addresses. That is, the 10.0.0.0/24 addresses can only have meaning within the given home network. But if private addresses only have meaning within a given network, how is addressing handled when packets are sent to or received from the global Internet, where addresses are necessarily unique? The answer lies in understanding NAT.

The NAT-enabled router does not *look* like a router to the outside world. Instead the NAT router behaves to the outside world as a *single* device with a *single* IP address. In Figure 4.22, all traffic leaving the home router for the larger Internet has a source IP address of 138.76.29.7, and all traffic entering the home router must have a destination address of 138.76.29.7. In essence, the NAT-enabled router is hiding the details of the home network from the outside world. (As an aside, you might wonder where the home network computers get their addresses and where the router gets its single IP address. Often, the answer is the same—DHCP! The router gets its address from the ISP's DHCP server, and the router runs a DHCP server to provide addresses to computers within the NAT-DHCP-router-controlled home network's address space.)

DELL - Exhibit 1008 - Page 137

If all datagrams arriving at the NAT router from the WAN have the same destination IP address (specifically, that of the WAN-side interface of the NAT router), then how does the router know the internal host to which it should forward a given datagram? The trick is to use a **NAT translation table** at the NAT router, and to include port numbers as well as IP addresses in the table entries.

Consider the example in Figure 4.22. Suppose a user sitting in a home network behind host 10.0.0.1 requests a Web page on some Web server (port 80) with IP address 128.119.40.186. The host 10.0.0.1 assigns the (arbitrary) source port number 3345 and sends the datagram into the LAN. The NAT router receives the datagram, generates a new source port number 5001 for the datagram, replaces the source IP address with its WAN-side IP address 138.76.29.7, and replaces the original source port number 3345 with the new source port number 5001. When generating a new source port number, the NAT router can select any source port number that is not currently in the NAT translation table. (Note that because a port number field is 16 bits long, the NAT protocol can support over 60,000 simultaneous connections with a single WAN-side IP address for the router!) NAT in the router also adds an entry to its NAT translation table. The Web server, blissfully unaware that the arriving datagram containing the HTTP request has been manipulated by the NAT router, responds with a datagram whose destination address is the IP address of the NAT router, and whose destination port number is 5001. When this datagram arrives at the NAT router, the router indexes the NAT translation table using the destination IP address and destination port number to obtain the appropriate IP address (10.0.0.1) and destination port number (3345) for the browser in the home network. The router then rewrites the datagram's destination address and destination port number, and forwards the datagram into the home network.

NAT has enjoyed widespread deployment in recent years. But we should mention that many purists in the IETF community loudly object to NAT. First, they argue, port numbers are meant to be used for addressing processes, not for addressing hosts. (This violation can indeed cause problems for servers running on the home network, since, as we have seen in Chapter 2, server processes wait for incoming requests at well-known port numbers.) Second, they argue, routers are supposed to process packets only up to layer 3. Third, they argue, the NAT protocol violates the so-called end-to-end argument; that is, hosts should be talking directly with each other, without interfering nodes modifying IP addresses and port numbers. And fourth, they argue, we should use IPv6 (see Section 4.4.4) to solve the shortage of IP addresses, rather than recklessly patching up the problem with a stopgap solution like NAT. But like it or not, NAT has become an important component of the Internet.

Yet another major problem with NAT is that it interferes with P2P applications, including P2P file-sharing applications and P2P Voice over IP applications. Recall from Chapter 2 that in a P2P application, any participating Peer A should be able to initiate a TCP connection to any other participating Peer B. The essence of the problem is that if Peer B is behind a NAT, it cannot act as a server and accept TCP

DELL - Exhibit 1008 - Page 138

connections. As we'll see in the homework problems, this NAT problem can be circumvented if Peer A is not behind a NAT. In this case, Peer A can first contact Peer B through an intermediate Peer C, which is not behind a NAT and to which B has established an ongoing TCP connection. Peer A can then ask Peer B, via Peer C, to initiate a TCP connection directly back to Peer A. Once the direct P2P TCP connection is established between Peers A and B, the two peers can exchange messages or files. This hack, called **connection reversal**, is actually used by many P2P applications for **NAT traversal**. If both Peer A and Peer B are behind their own NATs, the situation is a bit trickier but can be handled using application relays, as we saw with Skye relays in Chapter 2.

UPnP

NAT traversal is increasingly provided by Universal Plug and Play (UPnP), which is a protocol that allows a host to discover and configure a nearby NAT [UPnP Forum 2007]. UPnP requires that both the host and the NAT be UPnP compatible. With UPnP, an application running in a host can request a NAT mapping between its (*private IP address, private port number*) and the (*public IP address, public port number*) for some requested public port number. If the NAT accepts the request and creates the mapping, then nodes from the outside can initiate TCP connections to (*public IP address, public port number*). Furthermore, UPnP lets the application know the value of (*public IP address, public port number*), so that the application can advertise it to the outside world.

As an example, suppose your host, behind a UPnP-enabled NAT, has private address 10.0.0.1 and is running BitTorrent on port 3345. Also suppose that the public IP address of the NAT is 138.76.29.7. Your BitTorrent application naturally wants to be able to accept connections from other hosts, so that it can trade chunks with them. To this end, the BitTorrent application in your host asks the NAT to create a "hole" that maps (10.0.0.1, 3345) to (138.76.29.7, 5001). (The public port number 5001 is chosen by the application.) The BitTorrent application in your host could also advertise to its tracker that it is available at (138.76.29.7, 5001). In this manner, an external host running BitTorrent can contact the tracker and learn that your BitTorrent application is running at (138.76.29.7, 5001). The external host can send a TCP SYN packet to (138.76.29.7, 5001). When the NAT receives the SYN packet, it will change the destination IP address and port number in the packet to (10.0.0.1, 3345) and forward the packet through the NAT.

In summary, UPnP allows external hosts to initiate communication sessions to NATed hosts, using either TCP or UDP. NATs have long been a nemesis for P2P applications; UPnP, providing an effective and robust NAT traversal solution, may be their savior. Our discussion of NAT and UPnP here has been necessarily brief. For more detailed discussions of NAT see [Huston and UPnP 2004, Cisco NAT 2007].

DELL - Exhibit 1008 - Page 139

4.4.3 Internet Control Message Protocol (ICMP)

Recall that the network layer of the Internet has three main components: the IP protocol, discussed in the previous section; the Internet routing protocols (including RIP, OSPF, and BGP), which are covered in Section 4.6; and ICMP, which is the subject of this section.

ICMP, specified in [RFC 792], is used by hosts and routers to communicate network-layer information to each other. The most typical use of ICMP is for error reporting. For example, when running a Telnet, FTP, or HTTP session, you may have encountered an error message such as "Destination network unreachable." This message had its origins in ICMP. At some point, an IP router was unable to find a path to the host specified in your Telnet, FTP, or HTTP application. That router created and sent a type-3 ICMP message to your host indicating the error.

ICMP is often considered part of IP but architecturally it lies just above IP, as ICMP messages are carried inside IP datagrams. That is, ICMP messages are carried as IP payload, just as TCP or UDP segments are carried as IP payload. Similarly, when a host receives an IP datagram with ICMP specified as the upper-layer protocol, it demultiplexes the datagram's contents to ICMP, just as it would demultiplex a datagram's content to TCP or UDP.

ICMP messages have a type and a code field, and contain the header and the first 8 bytes of the IP datagram that caused the ICMP message to be generated in the first place (so that the sender can determine the datagram that caused the error). Selected ICMP message types are shown in Figure 4.23. Note that ICMP messages are used not only for signaling error conditions.

The well-known ping program sends an ICMP type 8 code 0 message to the specified host. The destination host, seeing the echo request, sends back a type 0 code 0 ICMP echo reply. Most TCP/IP implementations support the `ping` server directly in the operating system; that is, the server is not a process. Chapter 11 of [Stevens 1990] provides the source code for the `ping` client program. Note that the client program needs to be able to instruct the operating system to generate an ICMP message of type 8 code 0.

Another interesting ICMP message is the source quench message. This message is seldom used in practice. Its original purpose was to perform congestion control—to allow a congested router to send an ICMP source quench message to a host to force that host to reduce its transmission rate. We have seen in Chapter 3 that TCP has its own congestion-control mechanism that operates at the transport layer, without the use of network-layer feedback such as the ICMP source quench message.

In Chapter 1 we introduced the Traceroute program, which allows us to trace a route from a host to any other host in the world. Interestingly, Traceroute is implemented with ICMP messages. To determine the names and addresses of the routers between source and destination, Traceroute in the source sends a series of ordinary IP datagrams to the destination. Each of these datagrams carries a UDP segment with an unlikely UDP port number. The first of these datagrams has a TTL of 1, the

DELL - Exhibit 1008 - Page 140

ICMP Type	Code	Description
0	0	echo reply (to ping)
3	0	destination network unreachable
3	1	destination host unreachable
3	2	destination protocol unreachable
3	3	destination port unreachable
3	6	destination network unknown
3	7	destination host unknown
4	0	source quench (congestion control)
8	0	echo request
9	0	router advertisement
10	0	router discovery
11	0	TTL expired
12	0	IP header bad

Figure 4.23 ♦ ICMP message types

second of 2, the third of 3, and so on. The source also starts timers for each of the datagrams. When the *n*th datagram arrives at the *n*th router, the *n*th router observes that the TTL of the datagram has just expired. According to the rules of the IP protocol, the router discards the datagram and sends an ICMP warning message to the source (type 11 code 0). This warning message includes the name of the router and its IP address. When this ICMP message arrives back at the source, the source obtains the round-trip time from the timer and the name and IP address of the *n*th router from the ICMP message.

How does a Traceroute source know when to stop sending UDP segments? Recall that the source increments the TTL field for each datagram it sends. Thus, one of the datagrams will eventually make it all the way to the destination host. Because this datagram contains a UDP segment with an unlikely port number, the destination host sends a port unreachable ICMP message (type 3 code 3) back to the source. When the source host receives this particular ICMP message, it knows it does not need to send additional probe packets. (The standard Traceroute program actually sends sets of three packets with the same TTL; thus the Traceroute output provides three results for each TTL.)

DELL - Exhibit 1008 - Page 141

FOCUS ON SECURITY

INSPECTING DATAGRAMS: FIREWALLS AND INTRUSION DETECTION SYSTEMS

Suppose you are assigned the task of administering a home, departmental, university, or corporate network. Attackers, knowing the IP address range of your network, can easily send IP datagrams to addresses in your range. These datagrams can do all kinds of devious things, including mapping your network with ping sweeps and port scans, crashing vulnerable hosts with malformed packets, flooding servers with a deluge of ICMP packets, and infecting hosts by including malware in the packets. As the network administrator, what are you going to do about all those bad guys out there, each capable of sending malicious packets into your network? Two popular defense mechanisms to malicious packet attacks are firewalls and intrusion detection systems (IDSs).

As a network administrator, you may first try installing a firewall between your network and the Internet. (Most access routers today have firewall capability.) Firewalls inspect the datagram and segment header fields, denying suspicious datagrams entry into the internal network. For example, a firewall may be configured to block all ICMP echo request packets, thereby preventing an attacker from doing a traditional ping sweep across your IP address range. Firewalls can also block packets based on source and destination IP addresses and port numbers. Additionally, firewalls can be configured to track TCP connections, granting entry only to datagrams that belong to approved connections.

Additional protection can be provided with an IDS. An IDS, typically situated at the network boundary, performs "deep packet inspection," examining not only header fields but also the payloads in the datagram (including application-layer data). An IDS has a database of packet signatures that are known to be part of attacks. This database is automatically updated as new attacks are discovered. As packets pass through the IDS, the IDS attempts to match header fields and payloads to the signatures in its signature database. If such a match is found, an alert is created. An intrusion prevention systems (IPS) is similar to an IDS, except that it actually blocks packets in addition to creating alerts. In Chapter 8, we'll explore firewalls and IDSs in more detail.

Can firewalls and IDSs fully shield your network from all attacks? The answer is clearly no, as attackers continually find new attacks for which signatures are not yet available. But firewalls and traditional signature-based IDSs are useful in protecting your network from known attacks.

In this manner, the source host learns the number and the identities of routers that lie between it and the destination host and the round-trip time between the two hosts. Note that the Traceroute client program must be able to instruct the operating system to generate UDP datagrams with specific TTL values and must also be able to be notified by its operating system when ICMP messages arrive. Now that you understand how Traceroute works, you may want to go back and play with it some more.

DELL - Exhibit 1008 - Page 142

4.4.4 IPv6

In the early 1990s, the Internet Engineering Task Force began an effort to develop a successor to the IPv4 protocol. A prime motivation for this effort was the realization that the 32-bit IP address space was beginning to be used up, with new subnets and IP nodes being attached to the Internet (and being allocated unique IP addresses) at a breathtaking rate. To respond to this need for a large IP address space, a new IP protocol, IPv6, was developed. The designers of IPv6 also took this opportunity to tweak and augment other aspects of IPv4, based on the accumulated operational experience with IPv4.

The point in time when IPv4 addresses would be completely allocated (and hence no new subnets could attach to the Internet) was the subject of considerable debate. The estimates of the two leaders of the IETF's Address Lifetime Expectations working group were that addresses would become exhausted in 2008 and 2018, respectively [Solensky 1996]. In 1996, the American Registry for Internet Numbers (ARIN) reported that all of the IPv4 class A addresses had been assigned, 62 percent of the class B addresses had been assigned, and 37 percent of the class C addresses had been assigned [ARIN 1996]. For a recent report on IPv4 address space allocation, see [Hain 2005]. Although these estimates and numbers suggested that a considerable amount of time might be left until the IPv4 address space was exhausted, it was realized that considerable time would be needed to deploy a new technology on such an extensive scale, and so the Next Generation IP (IPng) effort [Bradner 1996; RFC 1752] was begun. The result of this effort was the specification of IP version 6 (IPv6) [RFC 2460]. (An often-asked question is what happened to IPv5. It was initially envisioned that the ST-2 protocol would become IPv5, but ST-2 was later dropped in favor of the RSVP protocol, which we'll discuss in Chapter 7.)

Excellent sources of information about IPv6 are The IP Next Generation Homepage [Hinden 2007] and a book on the subject by Huitema [Huitema 1998].

IPv6 Datagram Format

The format of the IPv6 datagram is shown in Figure 4.24. The most important changes introduced in IPv6 are evident in the datagram format:

- *Expanded addressing capabilities.* IPv6 increases the size of the IP address from 32 to 128 bits. This ensures that the world won't run out of IP addresses. Now, every grain of sand on the planet can be IP-addressable. In addition to unicast and multicast addresses, IPv6 has introduced a new type of address, called an **anycast address**, which allows a datagram to be delivered to any one of a group of hosts. (This feature could be used, for example, to send an HTTP GET to the nearest of a number of mirror sites that contain a given document.)
- *A streamlined 40-byte header.* As discussed below, a number of IPv4 fields have been dropped or made optional. The resulting 40-byte fixed-length header allows

DELL - Exhibit 1008 - Page 143

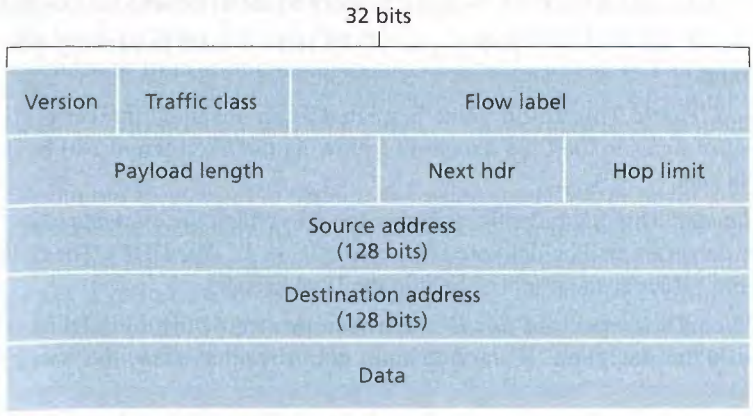


Figure 4.24 ♦ IPv6 datagram format

for faster processing of the IP datagram. A new encoding of options allows for more flexible options processing.

- *Flow labeling and priority.* IPv6 has an elusive definition of a **flow**. RFC 1752 and RFC 2460 state that this allows "labeling of packets belonging to particular flows for which the sender requests special handling, such as a nondefault quality of service or real-time service." For example, audio and video transmission might likely be treated as a flow. On the other hand, the more traditional applications, such as file transfer and e-mail, might not be treated as flows. It is possible that the traffic carried by a high-priority user (for example, someone paying for better service for their traffic) might also be treated as a flow. What is clear, however, is that the designers of IPv6 foresee the eventual need to be able to differentiate among the flows, even if the exact meaning of a flow has not yet been determined. The IPv6 header also has an 8-bit traffic class field. This field, like the TOS field in IPv4, can be used to give priority to certain datagrams within a flow, or it can be used to give priority to datagrams from certain applications (for example, ICMP) over datagrams from other applications (for example, network news).

As noted above, a comparison of Figure 4.24 with Figure 4.13 reveals the simpler, more streamlined structure of the IPv6 datagram. The following fields are defined in IPv6:

- *Version.* This 4-bit field identifies the IP version number. Not surprisingly, IPv6 carries a value of 6 in this field. Note that putting a 4 in this field does not create a valid IPv4 datagram. (If it did, life would be a lot simpler—see the discussion below regarding the transition from IPv4 to IPv6.)

DELL - Exhibit 1008 - Page 144

- *Traffic class.* This 8-bit field is similar in spirit to the TOS field we saw in IPv4.
- *Flow label.* As discussed above, this 20-bit field is used to identify a flow of datagrams.
- *Payload length.* This 16-bit value is treated as an unsigned integer giving the number of bytes in the IPv6 datagram following the fixed-length, 40-byte datagram header.
- *Next header.* This field identifies the protocol to which the contents (data field) of this datagram will be delivered (for example, to TCP or UDP). The field uses the same values as the protocol field in the IPv4 header.
- *Hop limit.* The contents of this field are decremented by one by each router that forwards the datagram. If the hop limit count reaches zero, the datagram is discarded.
- *Source and destination addresses.* The various formats of the IPv6 128-bit address are described in RFC 4291.
- *Data.* This is the payload portion of the IPv6 datagram. When the datagram reaches its destination, the payload will be removed from the IP datagram and passed on to the protocol specified in the next header field.

The discussion above identified the purpose of the fields that are included in the IPv6 datagram. Comparing the IPv6 datagram format in Figure 4.24 with the IPv4 datagram format that we saw in Figure 4.13, we notice that several fields appearing in the IPv4 datagram are no longer present in the IPv6 datagram:

- *Fragmentation/Reassembly.* IPv6 does not allow for fragmentation and reassembly at intermediate routers; these operations can be performed only by the source and destination. If an IPv6 datagram received by a router is too large to be forwarded over the outgoing link, the router simply drops the datagram and sends a "Packet Too Big" ICMP error message (see below) back to the sender. The sender can then resend the data, using a smaller IP datagram size. Fragmentation and reassembly is a time-consuming operation; removing this functionality from the routers and placing it squarely in the end systems considerably speeds up IP forwarding within the network.
- *Header checksum.* Because the transport-layer (for example, TCP and UDP) and data link–layer (for example, Ethernet) protocols in the Internet layers perform checksumming, the designers of IP probably felt that this functionality was sufficiently redundant in the network layer that it could be removed. Once again, fast processing of IP packets was a central concern. Recall from our discussion of IPv4 in Section 4.4.1 that since the IPv4 header contains a TTL field (similar to the hop limit field in IPv6), the IPv4 header checksum needed to be recomputed at every router. As with fragmentation and reassembly, this too was a costly operation in IPv4.

DELL - Exhibit 1008 - Page 145

- *Options.* An options field is no longer a part of the standard IP header. However, it has not gone away. Instead, the options field is one of the possible next headers pointed to from within the IPv6 header. That is, just as TCP or UDP protocol headers can be the next header within an IP packet, so too can an options field. The removal of the options field results in a fixed-length, 40-byte IP header.

Recall from our discussion in Section 4.4.3 that the ICMP protocol is used by IP nodes to report error conditions and provide limited information (for example, the echo reply to a ping message) to an end system. A new version of ICMP has been defined for IPv6 in RFC 4443. In addition to reorganizing the existing ICMP type and code definitions, ICMPv6 also added new types and codes required by the new IPv6 functionality. These include the "Packet Too Big" type, and an "unrecognized IPv6 options" error code. In addition, ICMPv6 subsumes the functionality of the Internet Group Management Protocol (IGMP) that we'll study in Section 4.7. IGMP, which is used to manage a host's joining and leaving of multicast groups, was previously a separate protocol from ICMP in IPv4.

Transitioning from IPv4 to IPv6

Now that we have seen the technical details of IPv6, let us consider a very practical matter: How will the public Internet, which is based on IPv4, be transitioned to IPv6? The problem is that while new IPv6-capable systems can be made backward-compatible, that is, can send, route, and receive IPv4 datagrams, already deployed IPv4-capable systems are not capable of handling IPv6 datagrams. Several options are possible.

One option would be to declare a flag day—a given time and date when all Internet machines would be turned off and upgraded from IPv4 to IPv6. The last major technology transition (from using NCP to using TCP for reliable transport service) occurred almost 25 years ago. Even back then [RFC 801], when the Internet was tiny and still being administered by a small number of "wizards," it was realized that such a flag day was not possible. A flag day involving hundreds of millions of machines and millions of network administrators and users is even more unthinkable today. RFC 4213 describes two approaches (which can be used either alone or together) for gradually integrating IPv6 hosts and routers into an IPv4 world (with the long-term goal, of course, of having all IPv4 nodes eventually transition to IPv6).

Probably the most straightforward way to introduce IPv6-capable nodes is a **dual-stack** approach, where IPv6 nodes also have a complete IPv4 implementation. Such a node, referred to as an IPv6/IPv4 node in RFC 4213, has the ability to send and receive both IPv4 and IPv6 datagrams. When interoperating with an IPv4 node, an IPv6/IPv4 node can use IPv4 datagrams; when interoperating with an IPv6 node, it can speak IPv6. IPv6/IPv4 nodes must have both IPv6 and IPv4 addresses. They must furthermore be able to determine whether another node is IPv6-capable or

DELL - Exhibit 1008 - Page 146

IPv4-only. This problem can be solved using the DNS (see Chapter 2), which can return an IPv6 address if the node name being resolved is IPv6-capable, or otherwise return an IPv4 address. Of course, if the node issuing the DNS request is only IPv4-capable, the DNS returns only an IPv4 address.

In the dual-stack approach, if either the sender or the receiver is only IPv4-capable, an IPv4 datagram must be used. As a result, it is possible that two IPv6-capable nodes can end up, in essence, sending IPv4 datagrams to each other. This is illustrated in Figure 4.25. Suppose Node A is IPv6-capable and wants to send an IP datagram to Node F, which is also IPv6-capable. Nodes A and B can exchange an IPv6 datagram. However, Node B must create an IPv4 datagram to send to C. Certainly, the data field of the IPv6 datagram can be copied into the data field of the IPv4 datagram and appropriate address mapping can be done. However, in performing the conversion from IPv6 to IPv4, there will be IPv6-specific fields in the IPv6 datagram (for example, the flow identifier field) that have no counterpart in IPv4. The information in these fields will be lost. Thus, even though E and F can exchange IPv6 datagrams, the arriving IPv4 datagrams at E from D do not contain all of the fields that were in the original IPv6 datagram sent from A.

An alternative to the dual-stack approach, also discussed in RFC 4213, is known as **tunneling**. Tunneling can solve the problem noted above, allowing, for example, E to receive the IPv6 datagram originated by A. The basic idea behind tunneling is the following. Suppose two IPv6 nodes (for example, B and E in Figure 4.25) want to interoperate using IPv6 datagrams but are connected to each other by intervening IPv4 routers. We refer to the intervening set of IPv4 routers between two IPv6 routers as a **tunnel**, as illustrated in Figure 4.26. With tunneling, the IPv6 node on the sending side of the tunnel (for example, B) takes the *entire* IPv6 datagram and puts it in the data (payload) field of an IPv4 datagram. This IPv4 datagram is then addressed to the IPv6 node on the receiving side of the tunnel (for example, E)

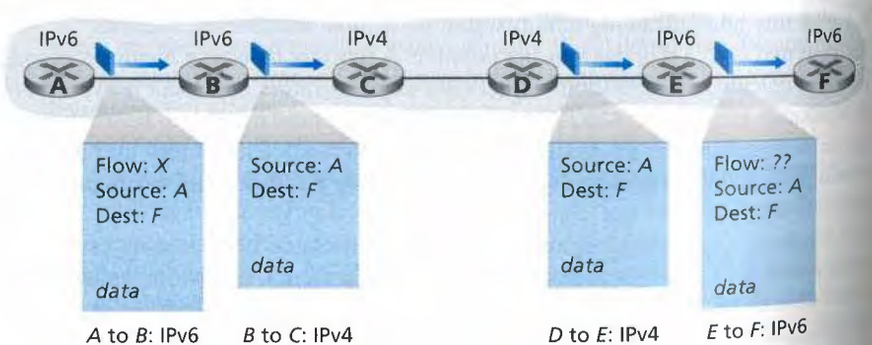


Figure 4.25 ♦ A dual-stack approach

DELL - Exhibit 1008 - Page 147

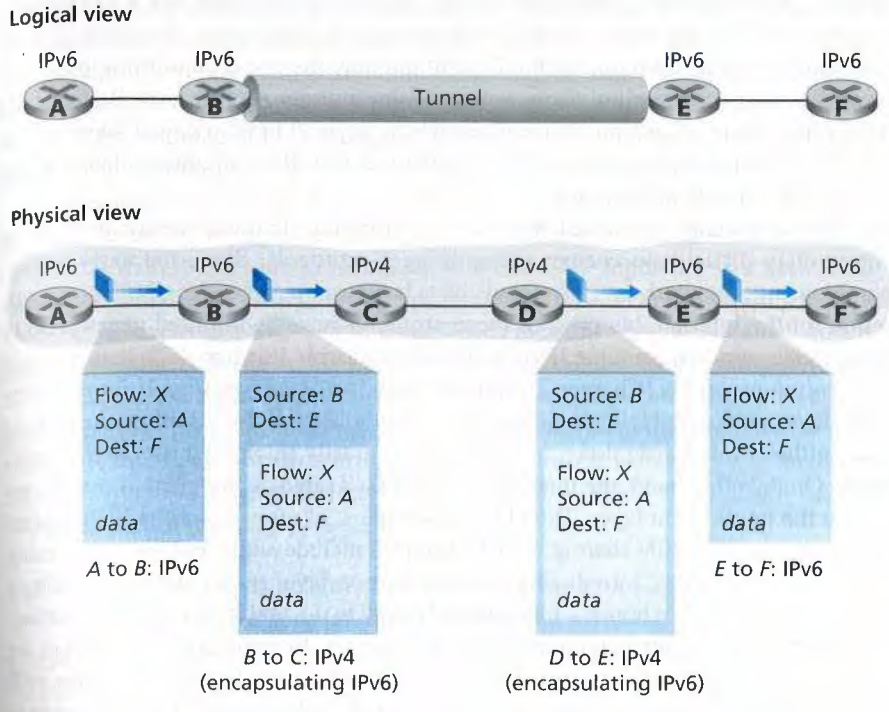


Figure 4.26 ♦ Tunneling

and sent to the first node in the tunnel (for example, C). The intervening IPv4 routers in the tunnel route this IPv4 datagram among themselves, just as they would any other datagram, blissfully unaware that the IPv4 datagram itself contains a complete IPv6 datagram. The IPv6 node on the receiving side of the tunnel eventually receives the IPv4 datagram (it is the destination of the IPv4 datagram!), determines that the IPv4 datagram contains an IPv6 datagram, extracts the IPv6 datagram, and then routes the IPv6 datagram exactly as it would if it had received the IPv6 datagram from a directly connected IPv6 neighbor.

We end this section by noting that while the adoption of IPv6 was initially slow to take off [Lawton 2001], momentum has been building recently. The US Office of Management and Budget (OMB) has required the transition to IPv6 by June 2008. The proliferation of devices such as IP-enabled phones and other portable devices provides an additional push for more widespread deployment of IPv6. Europe's Third Generation Partnership Program [3GPP 2007] has specified IPv6 as the standard addressing scheme for mobile multimedia. Even if IPv6 hasn't been widely deployed in the first 10 years of its young life, a long-term view is clearly called for.

DELL - Exhibit 1008 - Page 148

Today's phone number system took several decades to take hold, but it has been in place now for nearly half a century with no sign of going away. Similarly, it may take some time for IPv6 to take hold, but it too may then be around for a long time thereafter. Brian Carpenter, former chair of the Internet Architecture Board [IAB 2007] and author of several IPv6-related RFCs, says, "I have always looked at this as a 15-year process starting in 1995" [Lawton 2001]. By Carpenter's dates, we're nearing the three-quarters point!

One important lesson that we can learn from the IPv6 experience is that it is enormously difficult to change network-layer protocols. Since the early 1990s, numerous new network-layer protocols have been trumpeted as the next major revolution for the Internet, but most of these protocols have had limited penetration to date. These protocols include IPv6, multicast protocols (Section 4.7), and resource reservation protocols (Chapter 7). Indeed, introducing new protocols into the network layer is like replacing the foundation of a house—it is difficult to do without tearing the whole house down or at least temporarily relocating the house's residents. On the other hand, the Internet has witnessed rapid deployment of new protocols at the application layer. The classic examples, of course, are the Web, instant messaging, and P2P file sharing. Other examples include audio and video streaming and distributed games. Introducing new application-layer protocols is like adding a new layer of paint to a house—it is relatively easy to do, and if you choose an attractive color, others in the neighborhood will copy you. In summary, in the future we can expect to see changes in the Internet's network layer, but these changes will likely occur on a time scale that is much slower than the changes that will occur at the application layer.

4.4.5 A Brief Foray into IP Security

Section 4.4.3 covered IPv4 in some detail, including the services it provides and how those services are implemented. While reading through that section, you may have noticed that there was no mention of any security services. Indeed, IPv4 was designed in an era (the 1970s) when the Internet was primarily used among mutually-trusted networking researchers. Creating a computer network that integrated a multitude of link-layer technologies was already challenging enough, without having to worry about security.

But, with security being a major concern today, Internet researchers have moved on to design new network-layer protocols that provide a variety of security services. One of these protocols is IPsec, one of the more popular secure network-layer protocols and also widely deployed in Virtual Private Networks (VPNs). Although IPsec and its cryptographic underpinnings are covered in some detail in Chapter 8, we provide a brief, high-level introduction into IPsec services in this section.

DELL - Exhibit 1008 - Page 149

IPsec has been designed to be backward compatible with IPv4 and IPv6. In particular, in order to reap the benefits of IPsec, we don't need to replace the protocol stacks in *all* the routers and hosts in the Internet. For example, using the transport mode (one of two IPsec "modes"), if two hosts want to securely communicate, IPsec needs to be available only in those two hosts. All other routers and hosts can continue to run vanilla IPv4.

For concreteness, we'll focus on IPsec's transport mode here. In this mode, two hosts first establish an IPsec session between themselves. (Thus IPsec is connection-oriented!) With the session in place, all TCP and UDP segments sent between the two hosts enjoy the security services provided by IPsec. On the sending side, the transport layer passes a segment to IPsec. IPsec then encrypts the segment, appends additional security fields to the segment, and encapsulates the resulting payload in an ordinary IP datagram. (It's actually a little more complicated than this, as we'll see in Chapter 8.) The sending host then sends the datagram into the Internet, which transports it to the destination host. There, IPsec decrypts the segment and passes the unencrypted segment to the transport layer.

The services provided by an IPsec session include:

- *Cryptographic agreement*. Mechanisms that allow the two communicating hosts to agree on cryptographic algorithms and keys.
- *Encryption of IP datagram payloads*. When the sending host receives a segment from the transport layer, IPsec encrypts the payload. The payload can only be decrypted by IPsec in the receiving host.
- *Data integrity*. IPsec allows the receiving host to verify that the datagram's header fields and encrypted payload were not modified while the datagram was en route from source to destination.
- *Origin authentication*. When a host receives an IPsec datagram from a trusted source (with a trusted key—see Chapter 8), the host is assured that the source IP address in the datagram is the actual source of the datagram.

When two hosts have an IPsec session established between them, all TCP and UDP segments sent between them will be encrypted and authenticated. IPsec therefore provides blanket coverage, securing all communication between the two hosts for all network applications.

A company can use IPsec to communicate securely in the nonsecure public Internet. For illustrative purposes, we'll just look at a simple example here. Consider a company that has a large number of traveling salespeople, each possessing a company laptop computer. Suppose the salespeople need to frequently consult sensitive company information (for example, pricing and product information) that is stored on a server in the company's headquarters. Further suppose that the salespeople also need to send sensitive documents to each other. How can this be done with IPsec? As you

DELL - Exhibit 1008 - Page 150

might guess, we install IPsec in the server and in all of the salespeople's laptops. With IPsec installed in these hosts, whenever a salesperson needs to communicate with the server or with another salesperson, the communication session will be secure.

4.5 Routing Algorithms

So far in this chapter, we've mostly explored the network layer's forwarding function. We learned that when a packet arrives to a router, the router indexes a forwarding table and determines the link interface to which the packet is to be directed. We also learned that routing algorithms, operating in network routers, exchange and compute the information that is used to configure these forwarding tables. The interplay between routing algorithms and forwarding tables was shown in Figure 4.2. Having explored forwarding in some depth we now turn our attention to the other major topic of this chapter, namely, the network layer's critical routing function. Whether the network layer provides a datagram service (in which case different packets between a given source-destination pair may take different routes) or a VC service (in which case all packets between a given source and destination will take the same path), the network layer must nonetheless determine the path that packets take from senders to receivers. We'll see that the job of routing is to determine good paths (equivalently, routes), from senders to receivers, through the network of routers.

Typically a host is attached directly to one router, the **default router** for the host (also called the **first-hop router** for the host). Whenever a host sends a packet, the packet is transferred to its default router. We refer to the default router of the source host as the **source router** and the default router of the destination host as the **destination router**. The problem of routing a packet from source host to destination host clearly boils down to the problem of routing the packet from source router to destination router, which is the focus of this section.

The purpose of a routing algorithm is then simple: given a set of routers, with links connecting the routers, a routing algorithm finds a "good" path from source router to destination router. Typically, a good path is one that has the least cost. We'll see, however, that in practice, real-world concerns such as policy issues (for example, a rule such as "router *x*, belonging to organization *Y*, should not forward any packets originating from the network owned by organization *Z*") also come into play to complicate the conceptually simple and elegant algorithms whose theory underlies the practice of routing in today's networks.

A graph is used to formulate routing problems. Recall that a **graph** $G = (N,E)$ is a set N of nodes and a collection E of edges, where each edge is a pair of nodes from N. In the context of network-layer routing, the nodes in the graph represent routers—the points at which packet-forwarding decisions are made—and the edges

DELL - Exhibit 1008 - Page 151

connecting these nodes represent the physical links between these routers. Such a graph abstraction of a computer network is shown in Figure 4.27. To view some graphs representing real network maps, see [Dodge 2007, Cheswick 2000]; for a discussion of how well different graph-based models model the Internet, see [Zegura 1997, Faloutsos 1999, Li 2004].

As shown in Figure 4.27, an edge also has a value representing its cost. Typically, an edge's cost may reflect the physical length of the corresponding link (for example, a transoceanic link might have a higher cost than a short-haul terrestrial link), the link speed, or the monetary cost associated with a link. For our purposes, we'll simply take the edge costs as a given and won't worry about how they are determined. For any edge (x,y) in E, we denote $c(x,y)$ as the cost of the edge between nodes x and y. If the pair (x,y) does not belong to E, we set $c(x,y) = \infty$. Also, throughout we consider only undirected graphs (i.e., graphs whose edges do not have a direction), so that edge (x,y) is the same as edge (y,x) and that $c(x,y) = c(y,x)$. Also, a node y is said to be a **neighbor** of node x if (x,y) belongs to E.

Given that costs are assigned to the various edges in the graph abstraction, a natural goal of a routing algorithm is to identify the least costly paths between sources and destinations. To make this problem more precise, recall that a **path** in a graph $G = (N,E)$ is a sequence of nodes $(x_1, x_2,..., x_p)$ such that each of the pairs (x_1,x_2), $(x_2,x_3),...,(x_{p-1},x_p)$ are edges in E. The cost of a path $(x_1,x_2,..., x_p)$ is simply the sum of all the edge costs along the path, that is, $c(x_1,x_2) + c(x_2,x_3) + ... + c(x_{p-1},x_p)$. Given any two nodes x and y, there are typically many paths between the two nodes, with each path having a cost. One or more of these paths is a **least-cost path**. The least-cost problem is therefore clear: Find a path between the source and destination that has least cost. In Figure 4.27, for example, the least-cost path between source node u and destination node w is (u, x, y, w) with a path cost of 3. Note that if all edges in the

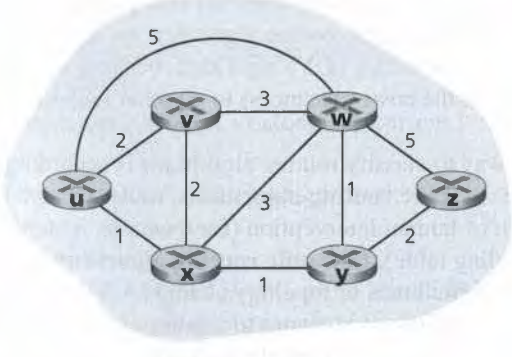


Figure 4.27 ♦ Abstract graph model of a computer network

DELL - Exhibit 1008 - Page 152

graph have the same cost, the least-cost path is also the **shortest path** (that is, the path with the smallest number of links between the source and the destination).

As a simple exercise, try finding the least-cost path from node u to z in Figure 4.27 and reflect for a moment on how you calculated that path. If you are like most people, you found the path from u to z by examining Figure 4.27, tracing a few routes from u to z, and somehow convincing yourself that the path you had chosen had the least cost among all possible paths. (Did you check all of the 17 possible paths between u and z? Probably not!) Such a calculation is an example of a centralized routing algorithm—the routing algorithm was run in one location, your brain, with complete information about the network. Broadly, one way in which we can classify routing algorithms is according to whether they are global or decentralized.

- A **global routing algorithm** computes the least-cost path between a source and destination using complete, global knowledge about the network. That is, the algorithm takes the connectivity between all nodes and all link costs as inputs. This then requires that the algorithm somehow obtain this information before actually performing the calculation. The calculation itself can be run at one site (a centralized global routing algorithm) or replicated at multiple sites. The key distinguishing feature here, however, is that a global algorithm has complete information about connectivity and link costs. In practice, algorithms with global state information are often referred to as **link-state (LS) algorithms**, since the algorithm must be aware of the cost of each link in the network. We'll study LS algorithms in Section 4.5.1.
- In a **decentralized routing algorithm**, the calculation of the least-cost path is carried out in an iterative, distributed manner. No node has complete information about the costs of all network links. Instead, each node begins with only the knowledge of the costs of its own directly attached links. Then, through an iterative process of calculation and exchange of information with its neighboring nodes (that is, nodes that are at the other end of links to which it itself is attached), a node gradually calculates the least-cost path to a destination or set of destinations. The decentralized routing algorithm we'll study below in Section 4.5.2 is called a distance-vector (DV) algorithm, because each node maintains a vector of estimates of the costs (distances) to all other nodes in the network.

A second broad way to classify routing algorithms is according to whether they are static or dynamic. In **static routing algorithms**, routes change very slowly over time, often as a result of human intervention (for example, a human manually editing a router's forwarding table). **Dynamic routing algorithms** change the routing paths as the network traffic loads or topology change. A dynamic algorithm can be run either periodically or in direct response to topology or link cost changes. While dynamic algorithms are more responsive to network changes, they are also more susceptible to problems such as routing loops and oscillation in routes.

DELL - Exhibit 1008 - Page 153

A third way to classify routing algorithms is according to whether they are load-sensitive or load-insensitive. In a **load-sensitive algorithm**, link costs vary dynamically to reflect the current level of congestion in the underlying link. If a high cost is associated with a link that is currently congested, a routing algorithm will tend to choose routes around such a congested link. While early ARPAnet routing algorithms were load-sensitive [McQuillan 1980], a number of difficulties were encountered [Huitema 1998]. Today's Internet routing algorithms (such as RIP, OSPF, and BGP) are **load-insensitive**, as a link's cost does not explicitly reflect its current (or recent past) level of congestion.

4.5.1 The Link-State (LS) Routing Algorithm

Recall that in a link-state algorithm, the network topology and all link costs are known, that is, available as input to the LS algorithm. In practice this is accomplished by having each node broadcast link-state packets to *all* other nodes in the network, with each link-state packet containing the identities and costs of its attached links. In practice (for example, with the Internet's OSPF routing protocol, discussed in Section 4.6.1) this is often accomplished by a **link-state broadcast** algorithm [Perlman 1999]. We'll cover broadcast algorithms in Section 4.7. The result of the nodes' broadcast is that all nodes have an identical and complete view of the network. Each node can then run the LS algorithm and compute the same set of least-cost paths as every other node.

The link-state routing algorithm we present below is known as *Dijkstra's algorithm*, named after its inventor. A closely related algorithm is Prim's algorithm; see [Cormen 2001] for a general discussion of graph algorithms. Dijkstra's algorithm computes the least-cost path from one node (the source, which we will refer to as u) to all other nodes in the network. Dijkstra's algorithm is iterative and has the property that after the kth iteration of the algorithm, the least-cost paths are known to k destination nodes, and among the least-cost paths to all destination nodes, these k paths will have the k smallest costs. Let us define the following notation:

- $D(v)$: cost of the least-cost path from the source node to destination v as of this iteration of the algorithm.
- $p(v)$: previous node (neighbor of v) along the current least-cost path from the source to v.
- N': subset of nodes; v is in N' if the least-cost path from the source to v is definitively known.

The global routing algorithm consists of an initialization step followed by a loop. The number of times the loop is executed is equal to the number of nodes in the network. Upon termination, the algorithm will have calculated the shortest paths from the source node u to every other node in the network.

DELL - Exhibit 1008 - Page 154

Link-State (LS) Algorithm for Source Node *u*

```
1  Initialization:
2    N' = {u}
3    for all nodes v
4      if v is a neighbor of u
5        then D(v) = c(u,v)
6      else D(v) = ∞
7
8  Loop
9    find w not in N' such that D(w) is a minimum
10   add w to N'
11   update D(v) for each neighbor v of w and not in N':
12         D(v) = min( D(v), D(w) + c(w,v) )
13   /* new cost to v is either old cost to v or known
14    least path cost to w plus cost from w to v */
15 until N'= N
```

As an example, let's consider the network in Figure 4.27 and compute the least-cost paths from *u* to all possible destinations. A tabular summary of the algorithm's computation is shown in Table 4.3, where each line in the table gives the values of the algorithm's variables at the end of the iteration. Let's consider the few first steps in detail.

- In the initialization step, the currently known least-cost paths from *u* to its directly attached neighbors, *v*, *x*, and *w*, are initialized to 2, 1, and 5, respectively. Note in particular that the cost to *w* is set to 5 (even though we will soon see that a lesser-cost path does indeed exist) since this is the cost of the direct (one hop) link from *u* to *w*. The costs to *y* and *z* are set to infinity because they are not directly connected to *u*.
- In the first iteration, we look among those nodes not yet added to the set N' and find that node with the least cost as of the end of the previous iteration. That node is *x*, with a cost of 1, and thus *x* is added to the set N'. Line 12 of the LS algorithm is then performed to update $D(v)$ for all nodes *v*, yielding the results shown in the second line (Step 1) in Table 4.3. The cost of the path to *v* is unchanged. The cost of the path to *w* (which was 5 at the end of the initialization) through node *x* is found to have a cost of 4. Hence this lower-cost path is selected and *w*'s predecessor along the shortest path from *u* is set to *x*. Similarly, the cost to *y* (through *x*) is computed to be 2, and the table is updated accordingly.
- In the second iteration, nodes *v* and *y* are found to have the least-cost paths (2), and we break the tie arbitrarily and add *y* to the set N' so that N' now contains *u*, *x*, and *y*. The cost to the remaining nodes not yet in N', that is, nodes *v*, *w*, and *z*,

DELL - Exhibit 1008 - Page 155

step	N′	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					

Table 4.3 ♦ Running the link-state algorithm on the network in Figure 4.27

are updated via line 12 of the LS algorithm, yielding the results shown in the third row in the Table 4.3.

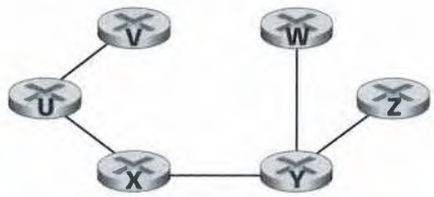
- And so on. . . .

When the LS algorithm terminates, we have, for each node, its predecessor along the least-cost path from the source node. For each predecessor, we also have *its* predecessor, and so in this manner we can construct the entire path from the source to all destinations. The forwarding table in a node, say node u, can then be constructed from this information by storing, for each destination, the next-hop node on the least-cost path from u to the destination. Figure 4.28 shows the resulting least-cost paths and forwarding table in u for the network in Figure 4.27.

What is the computational complexity of this algorithm? That is, given n nodes (not counting the source), how much computation must be done in the worst case to find the least-cost paths from the source to all destinations? In the first iteration, we need to search through all n nodes to determine the node, w, not in N' that has the minimum cost. In the second iteration, we need to check $n - 1$ nodes to determine the minimum cost; in the third iteration $n - 2$ nodes, and so on. Overall, the total number of nodes we need to search through over all the iterations is $n(n + 1)/2$, and thus we say that the preceding implementation of the LS algorithm has worst-case complexity of order n squared: $O(n^2)$. (A more sophisticated implementation of this algorithm, using a data structure known as a heap, can find the minimum in line 9 in logarithmic rather than linear time, thus reducing the complexity.)

Before completing our discussion of the LS algorithm, let us consider a pathology that can arise. Figure 4.29 shows a simple network topology where link costs are equal to the load carried on the link, for example, reflecting the delay that would be experienced. In this example, link costs are not symmetric; that is, $c(u,v)$ equals $c(v,u)$ only if the load carried on both directions on the link (u,v) is the same. In this example, node z originates a unit of traffic destined for w, node x also originates a

DELL - Exhibit 1008 - Page 156



Destination	Link
v	(u, v)
w	(u, x)
x	(u, x)
y	(u, x)
z	(u, x)

Figure 4.28 ♦ Least costs paths and forwarding table for nodule u

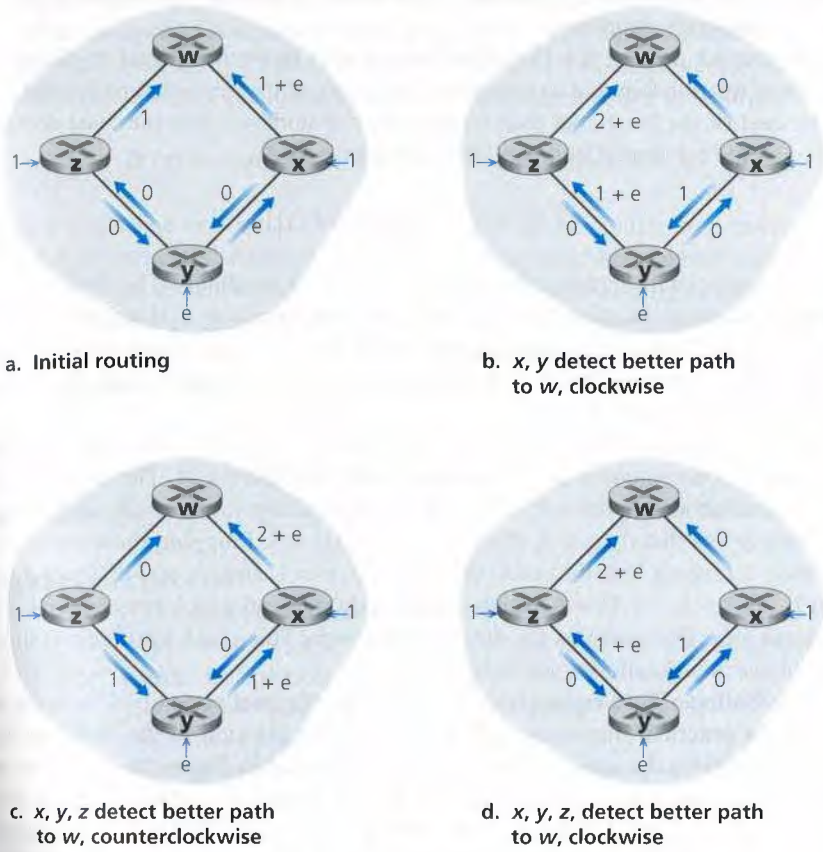
unit of traffic destined for *w*, and node *y* injects an amount of traffic equal to *e*, also destined for *w*. The initial routing is shown in Figure 4.29(a) with the link costs corresponding to the amount of traffic carried.

When the LS algorithm is next run, node *y* determines (based on the link costs shown in Figure 4.29(a)) that the clockwise path to *w* has a cost of 1, while the counterclockwise path to *w* (which it had been using) has a cost of 1 + *e*. Hence *y*'s least-cost path to *w* is now clockwise. Similarly, *x* determines that its new least-cost path to *w* is also clockwise, resulting in costs shown in Figure 4.29(b). When the LS algorithm is run next, nodes *x*, *y*, and *z* all detect a zero-cost path to *w* in the counterclockwise direction, and all route their traffic to the counterclockwise routes. The next time the LS algorithm is run, *x*, *y*, and *z* all then route their traffic to the clockwise routes.

What can be done to prevent such oscillations (which can occur in any algorithm, not just an LS algorithm, that uses a congestion or delay-based link metric)? One solution would be to mandate that link costs not depend on the amount of traffic carried—an unacceptable solution since one goal of routing is to avoid highly congested (for example, high-delay) links. Another solution is to ensure that not all routers run the LS algorithm at the same time. This seems a more reasonable solution, since we would hope that even if routers ran the LS algorithm with the same periodicity, the execution instance of the algorithm would not be the same at each node. Interestingly, researchers have found that routers in the Internet can self-synchronize among themselves [Floyd Synchronization 1994]. That is, even though they initially execute the algorithm with the same period but at different instants of time, the algorithm execution instance can eventually become, and remain, synchronized at the routers. One way to avoid such self-synchronization is for each router to randomize the time it sends out a link advertisement.

Having studied the LS algorithm, let's consider the other major routing algorithm that is used in practice today—the distance-vector routing algorithm.

DELL - Exhibit 1008 - Page 157



a. Initial routing

b. *x*, *y* detect better path to *w*, clockwise

c. *x*, *y*, *z* detect better path to *w*, counterclockwise

d. *x*, *y*, *z*, detect better path to *w*, clockwise

Figure 4.29 ♦ Oscillations with congestion-sensitive routing

4.5.2 The Distance-Vector (DV) Routing Algorithm

Whereas the LS algorithm is an algorithm using global information, the **distance-vector (DV)** algorithm is iterative, asynchronous, and distributed. It is *distributed* in that each node receives some information from one or more of its *directly attached* neighbors, performs a calculation, and then distributes the results of its calculation back to its neighbors. It is *iterative* in that this process continues on until no more information is exchanged between neighbors. (Interestingly, the algorithm is also self-terminating—there is no signal that the computation should stop; it just stops.) The algorithm is *asynchronous* in that it does not require all of the nodes to operate in lockstep with each other. We'll see that an asynchronous,

DELL - Exhibit 1008 - Page 158

iterative, self-terminating, distributed algorithm is much more interesting and fun than a centralized algorithm!

Before we present the DV algorithm, it will prove beneficial to discuss an important relationship that exists among the costs of the least-cost paths. Let $d_x(y)$ be the cost of the least-cost path from node x to node y. Then the least costs are related by the celebrated Bellman-Ford equation, namely,

$$d_x(y) = \min_v \{c(x,v) + d_v(y)\}, \tag{4.1}$$

where the min_v in the equation is taken over all of x's neighbors. The Bellman-Ford equation is rather intuitive. Indeed, after traveling from x to v, if we then take the least-cost path from v to y, the path cost will be $c(x,v) + d_v(y)$. Since we must begin by traveling to some neighbor v, the least cost from x to y is the minimum of $c(x,v) + d_v(y)$ taken over all neighbors v.

But for those who might be skeptical about the validity of the equation, let's check it for source node u and destination node z in Figure 4.27. The source node u has three neighbors: nodes v, x, and w. By walking along various paths in the graph, it is easy to see that $d_v(z) = 5$, $d_x(z) = 3$, and $d_w(z) = 3$. Plugging these values into Equation 4.1, along with the costs $c(u,v) = 2$, $c(u,x) = 1$, and $c(u,w) = 5$, gives $d_u(z) = \min\{2 + 5, 5 + 3, 1 + 3\} = 4$, which is obviously true and which is exactly what the Dijskstra algorithm gave us for the same network. This quick verification should help relieve any skepticism you may have.

The Bellman-Ford equation is not just an intellectual curiosity. It actually has significant practical importance. In particular, the solution to the Bellman-Ford equation provides the entries in node x's forwarding table. To see this, let v^* be any neighboring node that achieves the minimum in Equation 4.1. Then, if node x wants to send a packet to node y along a least-cost path, it should first forward the packet to node v^*. Thus, node x's forwarding table would specify node v^* as the next-hop router for the ultimate destination y. Another important practical contribution of the Bellman-Ford equation is that it suggests the form of the neighbor-to-neighbor communication that will take place in the DV algorithm.

The basic idea is as follows. Each node x begins with $D_x(y)$, an estimate of the cost of the least-cost path from itself to node y, for all nodes in N. Let $\mathbf{D}_x = [D_x(y): y \text{ in } N]$ be node x's distance vector, which is the vector of cost estimates from x to all other nodes, y, in N. With the DV algorithm, each node x maintains the following routing information:

- For each neighbor v, the cost $c(x,v)$ from x to directly attached neighbor, v
- Node x's distance vector, that is, $\mathbf{D}_x = [D_x(y): y \text{ in } N]$, containing x's estimate of its cost to all destinations, y, in N
- The distance vectors of each of its neighbors, that is, $\mathbf{D}_v = [D_v(y): y \text{ in } N]$ for each neighbor v of x

DELL - Exhibit 1008 - Page 159

In the distributed, asynchronous algorithm, from time to time, each node sends a copy of its distance vector to each of its neighbors. When a node x receives a new distance vector from any of its neighbors v, it saves v's distance vector, and then uses the Bellman-Ford equation to update its own distance vector as follows:

$$D_x(y) = \min_v\{c(x,v) + D_v(y)\} \quad \text{for each node } y \text{ in } N$$

If node x's distance vector has changed as a result of this update step, node x will then send its updated distance vector to each of its neighbors, which can in turn update their own distance vectors. Miraculously enough, as long as all the nodes continue to exchange their distance vectors in an asynchronous fashion, each cost estimate $D_x(y)$ converges to $d_x(y)$, the actual cost of the least-cost path from node x to node y [Bertsekas 1991]!

Distance-Vector (DV) Algorithm

At each node, x:

```
1  Initialization:
2     for all destinations y in N:
3        Dx(y) = c(x,y)    /* if y is not a neighbor then c(x,y) = ∞ */
4     for each neighbor w
5        Dw(y) = ∞ for all destinations y in N
6     for each neighbor w
7        send distance vector Dx = [Dx(y): y in N] to w
8
9  loop
10    wait (until I see a link cost change to some neighbor w or
11          until I receive a distance vector from some neighbor w)
12
13    for each y in N:
14       Dx(y) = minv{c(x,v) + Dv(y)}
15
16    if Dx(y) changed for any destination y
17       send distance vector Dx = [Dx(y): y in N] to all neighbors
18
19 forever
```

In the DV algorithm, a node x updates its distance-vector estimate when it either sees a cost change in one of its directly attached links or receives a distance-vector update from some neighbor. But to update its own forwarding table for a given destination y, what node x really needs to know is not the shortest-path dis-

DELL - Exhibit 1008 - Page 160

tance to y but instead the neighboring node $v^*(y)$ that is the next-hop router along the shortest path to y. As you might expect, the next-hop router $v^*(y)$ is the neighbor v that achieves the minimum in Line 14 of the DV algorithm. (If there are multiple neighbors v that achieve the minimum, then $v^*(y)$ can be any of the minimizing neighbors.) Thus, in Lines 13-14, for each destination y, node x also determines $v^*(y)$ and updates its forwarding table for destination y.

Recall that the LS algorithm is a global algorithm in the sense that it requires each node to first obtain a complete map of the network before running the Dijkstra algorithm. The DV algorithm is *decentralized* and does not use such global information. Indeed, the only information a node will have is the costs of the links to its directly attached neighbors and information it receives from these neighbors. Each node waits for an update from any neighbor (Lines 10–11), calculates its new distance vector when receiving an update (Line 14), and distributes its new distance vector to its neighbors (Lines 16–17). DV-like algorithms are used in many routing protocols in practice, including the Internet's RIP and BGP, ISO IDRP, Novell IPX, and the original ARPAnet.

Figure 4.30 illustrates the operation of the DV algorithm for the simple three-node network shown at the top of the figure. The operation of the algorithm is illustrated in a synchronous manner, where all nodes simultaneously receive distance vectors from their neighbors, compute their new distance vectors, and inform their neighbors if their distance vectors have changed. After studying this example, you should convince yourself that the algorithm operates correctly in an asynchronous manner as well, with node computations and update generation/reception occurring at any time.

The leftmost column of the figure displays three initial **routing tables** for each of the three nodes. For example, the table in the upper-left corner is node x's initial routing table. Within a specific routing table, each row is a distance vector—specifically, each node's routing table includes its own distance vector and that of each of its neighbors. Thus, the first row in node x's initial routing table is $\boldsymbol{D}_x = [D_x(x), D_x(y), D_x(z)] = [0, 2, 7]$. The second and third rows in this table are the most recently received distance vectors from nodes y and z, respectively. Because at initialization node x has not received anything from node y or z, the entries in the second and third rows are initialized to infinity.

After initialization, each node sends its distance vector to each of its two neighbors. This is illustrated in Figure 4.30 by the arrows from the first column of tables to the second column of tables. For example, node x sends its distance vector $\boldsymbol{D}_x = [0, 2, 7]$ to both nodes y and z. After receiving the updates, each node recomputes its own distance vector. For example, node x computes

$$D_x(x) = 0$$

$$D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\} = \min\{2 + 0, 7 + 1\} = 2$$

$$D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\} = \min\{2 + 1, 7 + 0\} = 3$$

DELL - Exhibit 1008 - Page 161

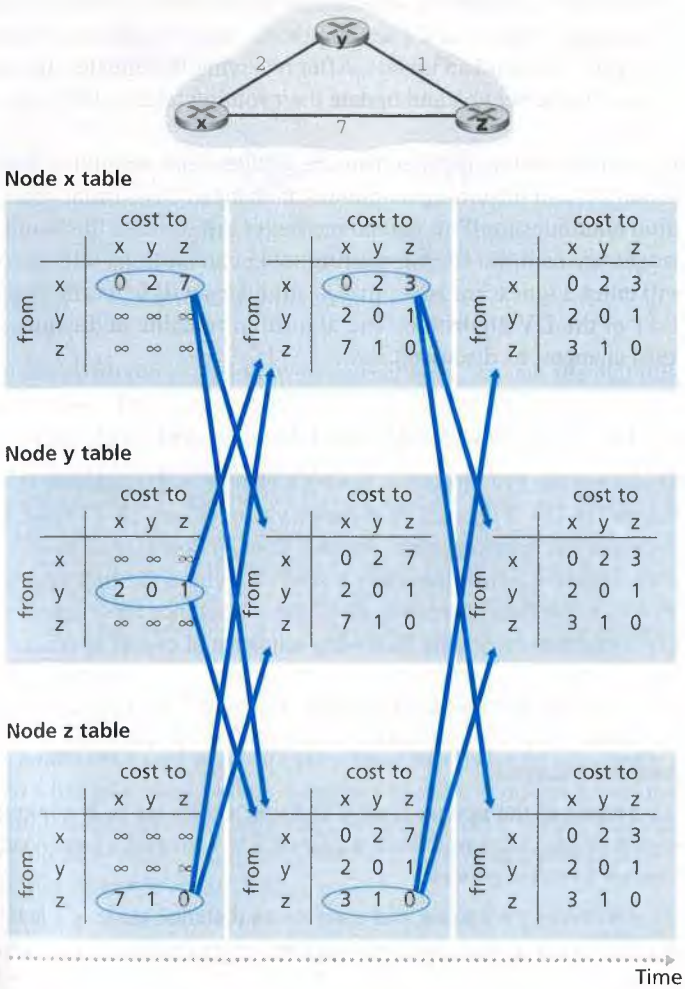


Figure 4.30 ♦ Distance-vector (DV) algorithm

The second column therefore displays, for each node, the node's new distance vector along with distance vectors just received from its neighbors. Note, for example, that node x's estimate for the least cost to node z, $D_x(z)$, has changed from 7 to 3. Also note that for node x, neighboring node y achieves the minimum in line 14 of the DV algorithm; thus at this stage of the algorithm, we have at node x that $v^*(y) = y$ and $v^*(z) = y$.

After the nodes recompute their distance vectors, they again send their updated distance vectors to their neighbors (if there has been a change). This is illustrated in

DELL - Exhibit 1008 - Page 162

Figure 4.30 by the arrows from the second column of tables to the third column of tables. Note that only nodes x and z send updates: node y's distance vector didn't change so node y doesn't send an update. After receiving the updates, the nodes then recompute their distance vectors and update their routing tables, which are shown in the third column.

The process of receiving updated distance vectors from neighbors, recomputing routing table entries, and informing neighbors of changed costs of the least-cost path to a destination continues until no update messages are sent. At this point, since no update messages are sent, no further routing table calculations will occur and the algorithm will enter a quiescent state; that is, all nodes will be performing the wait in Lines 10–11 of the DV algorithm. The algorithm remains in the quiescent state until a link cost changes, as discussed next.

Distance-Vector Algorithm: Link-Cost Changes and Link Failure

When a node running the DV algorithm detects a change in the link cost from itself to a neighbor (Lines 10–11), it updates its distance vector (Lines 13–14) and, if there's a change in the cost of the least-cost path, informs its neighbors (Lines 16–17) of its new distance vector. Figure 4.31(a) illustrates a scenario where the link cost from y to x changes from 4 to 1. We focus here only on y' and z's distance table entries to destination x. The DV algorithm causes the following sequence of events to occur:

- At time t_0, y detects the link-cost change (the cost has changed from 4 to 1), updates its distance vector, and informs its neighbors of this change since its distance vector has changed.
- At time t_1, z receives the update from y and updates its table. It computes a new least cost to x (it has decreased from a cost of 5 to a cost of 2) and sends its new distance vector to its neighbors.
- At time t_2, y receives z's update and updates its distance table. y's least costs do not change and hence y does not send any message to z. The algorithm comes to a quiescent state.

Thus, only two iterations are required for the DV algorithm to reach a quiescent state. The good news about the decreased cost between x and y has propagated quickly through the network.

Let's now consider what can happen when a link cost *increases*. Suppose that the link cost between x and y increases from 4 to 60, as shown in Figure 4.31(b).

1. Before the link cost changes, $D_y(x) = 4$, $D_y(z) = 1$, $D_z(y) = 1$, and $D_z(x) = 5$. At time t_0, y detects the link-cost change (the cost has changed from 4 to 60). y computes its new minimum-cost path to x to have a cost of

$$D_y(x) = \min\{c(y,x) + D_x(x),\ c(y,z) + D_z(x)\} = \min\{60 + 0,\ 1 + 5\} = 6$$

DELL - Exhibit 1008 - Page 163



a. b.

Figure 4.31 ♦ Changes in link cost

Of course, with our global view of the network, we can see that this new cost via z is *wrong*. But the only information node y has is that its direct cost to x is 60 and that z has last told y that z could get to x with a cost of 5. So in order to get to x, y would now route through z, fully expecting that z will be able to get to x with a cost of 5. As of t_1 we have a **routing loop**—in order to get to x, y routes through z, and z routes through y. A routing loop is like a black hole—a packet destined for x arriving at y or z as of t_1 will bounce back and forth between these two nodes forever (or until the forwarding tables are changed).

2. Since node y has computed a new minimum cost to x, it informs z of its new distance vector at time t_1.
3. Sometime after t_1, z receives y's new distance vector, which indicates that y's minimum cost to x is 6. z knows it can get to y with a cost of 1 and hence computes a new least cost to x of $D_z(x) = \min\{50 + 0, 1 + 6\} = 7$. Since z's least cost to x has increased, it then informs y of its new distance vector at t_2.
4. In a similar manner, after receiving z's new distance vector, y determines $D_y(x) = 8$ and sends z its distance vector. z then determines $D_z(x) = 9$ and sends y its distance vector, and so on.

How long will the process continue? You should convince yourself that the loop will persist for 44 iterations (message exchanges between y and z)—until z eventually computes the cost of its path via y to be greater than 50. At this point, z will (finally!) determine that its least-cost path to x is via its direct connection to x. y will then route to x via z. The result of the bad news about the increase in link cost has indeed traveled slowly! What would have happened if the link cost $c(y,x)$ had changed from 4 to 10,000 and the cost $c(z,x)$ had been 9,999? Because of such scenarios, the problem we have seen is sometimes referred to as the count-to-infinity problem.

Distance-Vector Algorithm: Adding Poisoned Reverse

The specific looping scenario just described can be avoided using a technique known as *poisoned reverse*. The idea is simple—if z routes through y to get to destination x, then z will advertise to y that its distance to x is infinity, that is, z will

DELL - Exhibit 1008 - Page 164

advertise to y that $D_z(x) = \infty$ (even though z knows $D_z(x) = 5$ in truth). z will continue telling this little white lie to y as long as it routes to x via y. Since y believes that z has no path to x, y will never attempt to route to x via z, as long as z continues to route to x via y (and lies about doing so).

Let's now see how poisoned reverse solves the particular looping problem we encountered before in Figure 4.31(b). As a result of the poisoned reverse, y's distance table indicates $D_z(x) = \infty$. When the cost of the (x,y) link changes from 4 to 60 at time t_0, y updates its table and continues to route directly to x, albeit at a higher cost of 60, and informs z of its new cost to x, that is, $D_y(x) = 60$. After receiving the update at t_1, z immediately shifts its route to x to be via the direct (z,x) link at a cost of 50. Since this is a new least-cost path to x, and since the path no longer passes through y, z now informs y that $D_z(x) = 50$ at t_2. After receiving the update from z, y updates its distance table with $D_y(x) = 51$. Also, since z is now on y's least-cost path to x, y poisons the reverse path from z to x by informing z at time t_3 that $D_y(x) = \infty$ (even though y knows that $D_y(x) = 51$ in truth).

Does poisoned reverse solve the general count-to-infinity problem? It does not. You should convince yourself that loops involving three or more nodes (rather than simply two immediately neighboring nodes) will not be detected by the poisoned reverse technique.

A Comparison of LS and DV Routing Algorithms

The DV and LS algorithms take complementary approaches towards computing routing. In the DV algorithm, each node talks to *only* its directly connected neighbors, but it provides its neighbors with least-cost estimates from itself to *all* the nodes (that it knows about) in the network. In the LS algorithm, each node talks with *all* other nodes (via broadcast), but it tells them *only* the costs of its directly connected links. Let's conclude our study of LS and DV algorithms with a quick comparison of some of their attributes. Recall that N is the set of nodes (routers) and E is the set of edges (links).

- *Message complexity.* We have seen that LS requires each node to know the cost of each link in the network. This requires $O(|N|\,|E|)$ messages to be sent. Also, whenever a link cost changes, the new link cost must be sent to all nodes. The DV algorithm requires message exchanges between directly connected neighbors at each iteration. We have seen that the time needed for the algorithm to converge can depend on many factors. When link costs change, the DV algorithm will propagate the results of the changed link cost only if the new link cost results in a changed least-cost path for one of the nodes attached to that link.
- *Speed of convergence.* We have seen that our implementation of LS is an $O(|N|^2)$ algorithm requiring $O(|N|\,|E|)$ messages. The DV algorithm can converge slowly and can have routing loops while the algorithm is converging. DV also suffers from the count-to-infinity problem.

DELL - Exhibit 1008 - Page 165

- *Robustness.* What can happen if a router fails, misbehaves, or is sabotaged? Under LS, a router could broadcast an incorrect cost for one of its attached links (but no others). A node could also corrupt or drop any packets it received as part of an LS broadcast. But an LS node is computing only its own forwarding tables; other nodes are performing similar calculations for themselves. This means route calculations are somewhat separated under LS, providing a degree of robustness. Under DV, a node can advertise incorrect least-cost paths to any or all destinations. (Indeed, in 1997, a malfunctioning router in a small ISP provided national backbone routers with erroneous routing information. This caused other routers to flood the malfunctioning router with traffic and caused large portions of the Internet to become disconnected for up to several hours [Neumann 1997].) More generally, we note that, at each iteration, a node's calculation in DV is passed on to its neighbor and then indirectly to its neighbor's neighbor on the next iteration. In this sense, an incorrect node calculation can be diffused through the entire network under DV.

In the end, neither algorithm is an obvious winner over the other; indeed, both algorithms are used in the Internet.

Other Routing Algorithms

The LS and DV algorithms we have studied are not only widely used in practice, they are essentially the *only* routing algorithms used in practice today in the Internet. Nonetheless, many routing algorithms have been proposed by researchers over the past 30 years, ranging from the extremely simple to the very sophisticated and complex. A broad class of routing algorithms is based on viewing packet traffic as flows between sources and destinations in a network. In this approach, the routing problem can be formulated mathematically as a constrained optimization problem known as a network flow problem [Bertsekas 1991]. Yet another set of routing algorithms we mention here are those derived from the telephony world. These **circuit-switched routing algorithms** are of interest to packet-switched data networking in cases where per-link resources (for example, buffers, or a fraction of the link bandwidth) are to be reserved for each connection that is routed over the link. While the formulation of the routing problem might appear quite different from the least-cost routing formulation we have seen in this chapter, there are a number of similarities, at least as far as the path-finding algorithm (routing algorithm) is concerned. See [Ash 1998; Ross 1995; Girard 1990] for a detailed discussion of this research area.

4.5.3 Hierarchical Routing

In our study of LS and DV algorithms, we've viewed the network simply as a collection of interconnected routers. One router was indistinguishable from another in

DELL - Exhibit 1008 - Page 166

the sense that all routers executed the same routing algorithm to compute routing paths through the entire network. In practice, this model and its view of a homogenous set of routers all executing the same routing algorithm is a bit simplistic for at least two important reasons:

- *Scale.* As the number of routers becomes large, the overhead involved in computing, storing, and communicating routing information (for example, LS updates or least-cost path changes) becomes prohibitive. Today's public Internet consists of hundreds of millions of hosts. Storing routing information at each of these hosts would clearly require enormous amounts of memory. The overhead required to broadcast LS updates among all of the routers in the public Internet would leave no bandwidth left for sending data packets! A distance-vector algorithm that iterated among such a large number of routers would surely never converge. Clearly, something must be done to reduce the complexity of route computation in networks as large as the public Internet.
- *Administrative autonomy.* Although researchers tend to ignore issues such as a company's desire to run its routers as it pleases (for example, to run whatever routing algorithm it chooses) or to hide aspects of its network's internal organization from the outside, these are important considerations. Ideally, an organization should be able to run and administer its network as it wishes, while still being able to connect its network to other outside networks.

Both of these problems can be solved by organizing routers into **autonomous systems** (**ASs**), with each AS consisting of a group of routers that are typically under the same administrative control (e.g., operated by the same ISP or belonging to the same company network). Routers within the same AS all run the same routing algorithm (for example, an LS or DV algorithm) and have information about each other—exactly as was the case in our idealized model in the preceding section. The routing algorithm running within an autonomous system is called an **intra-autonomous system routing protocol**. It will be necessary, of course, to connect ASs to each other, and thus one or more of the routers in an AS will have the added task of being responsible for forwarding packets to destinations outside the AS; these routers are called **gateway routers**.

Figure 4.32 provides a simple example with three ASs: AS1, AS2, and AS3. In this figure, the heavy lines represent direct link connections between pairs of routers. The thinner lines hanging from the routers represent subnets that are directly connected to the routers. AS1 has four routers—1a, 1b, 1c, and 1d—which run the intra-AS routing protocol used within AS1. Thus, each of these four routers knows how to forward packets along the optimal path to any destination within AS1. Similarly, autonomous systems AS2 and AS3 each have three routers. Note that the intra-AS routing protocols running in AS1, AS2, and AS3 need not be the same. Also note that the routers 1b, 1c, 2a, and 3a are all gateway routers.

DELL - Exhibit 1008 - Page 167

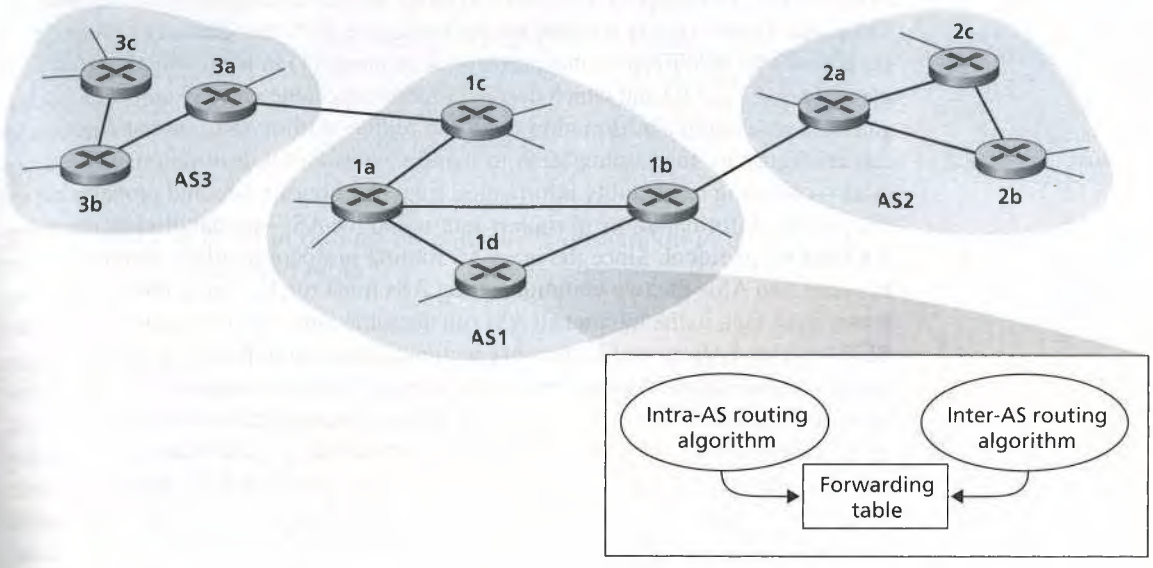


Figure 4.32 ♦ An example of interconnected autonomous systems

It should now be clear how the routers in an AS determine routing paths for source-destination pairs that are internal to the AS. But there is still a big missing piece to the end-to-end routing puzzle. How does a router, within some AS, know how to route a packet to a destination that is outside the AS? It's easy to answer this question if the AS has only one gateway router that connects to only one other AS. In this case, because the AS's intra-AS routing algorithm has determined the least-cost path from each internal router to the gateway router, each internal router knows how it should forward the packet. The gateway router, upon receiving the packet, forwards the packet on the one link that leads outside the AS. The AS on the other side of the link then takes over the responsibility of routing the packet to its ultimate destination. As an example, suppose router 2b in Figure 4.32 receives a packet whose destination is outside of AS2. Router 2b will then forward the packet to either router 2a or 2c, as specified by router 2b's forwarding table, which was configured by AS2's intra-AS routing protocol. The packet will eventually arrive to the gateway router 2a, which will forward the packet to 1b. Once the packet has left 2a, AS2's job is done with this one packet.

So the problem is easy when the source AS has only one link that leads outside the AS. But what if the source AS has two or more links (through two or more gateway routers) that lead outside the AS? Then the problem of knowing where to forward the packet becomes significantly more challenging. For example, consider a

DELL - Exhibit 1008 - Page 168

router in AS1 and suppose it receives a packet whose destination is outside the AS. The router should clearly forward the packet to one of its two gateway routers, 1b or 1c, but which one? To solve this problem, AS1 needs (1) to learn which destinations are reachable via AS2 and which destinations are reachable via AS3 and (2) to propagate this reachability information to all the routers within AS1, so that each router can configure its forwarding table to handle external-AS destinations. These two tasks—obtaining reachability information from neighboring ASs and propagating the reachability information to all routers internal to the AS—are handled by the **inter-AS routing protocol**. Since the inter-AS routing protocol involves communication between two ASs, the two communicating ASs must run the same inter-AS routing protocol. In fact, in the Internet all ASs run the same inter-AS routing protocol, called BGP4, which is discussed in the next section. As shown in Figure 4.32, each router receives information from an intra-AS routing protocol and an inter-AS routing protocol, and uses the information from both protocols to configure its forwarding table.

As an example, consider a subnet x (identified by its CIDRized address), and suppose that AS1 learns from the inter-AS routing protocol that subnet x is reachable from AS3 but is *not* reachable from AS2. AS1 then propagates this information to all of its routers. When router 1d learns that subnet x is reachable from AS3, and hence from gateway 1c, it then determines, from the information provided by the intra-AS routing protocol, the router interface that is on the least-cost path from router 1d to gateway router 1c. Say this is interface I. The router 1d can then put the entry (x, I) into its forwarding table. (This example, and others presented in this section, gets the general ideas across but is a simplification of what really happens in the Internet. In the next section we'll provide a more detailed description, albeit more complicated, when we discuss BGP.)

Following up on the previous example, now suppose that AS2 and AS3 connect to other ASs, which are not shown in the diagram. Also suppose that AS1 learns from the inter-AS routing protocol that subnet x is reachable both from AS2, via gateway 1b, and from AS3, via gateway 1c. AS1 would then propagate this information to all its routers, including router 1d. In order to configure its forwarding table, router 1d would have to determine to which gateway router, 1b or 1c, it should direct packets that are destined for subnet x. One approach, which is often employed in practice, is to use **hot-potato routing**. In hot-potato routing, the AS gets rid of the packet (the hot potato) as quickly as possible (more precisely, as inexpensively as possible). This is done by having a router send the packet to the gateway router that has the smallest router-to-gateway cost among all gateways with a path to the destination. In the context of the current example, hot-potato routing, running in 1d, would use information from the intra-AS routing protocol to determine the path costs to 1b and 1c, and then choose the path with the least cost. Once this path is chosen, router 1d adds an entry for subnet x in its forwarding table. Figure 4.33 summarizes the actions taken at router 1d for adding the new entry for x to the forwarding table.

When an AS learns about a destination from a neighboring AS, the AS can advertise this routing information to some of its other neighboring ASs. For example,

DELL - Exhibit 1008 - Page 169

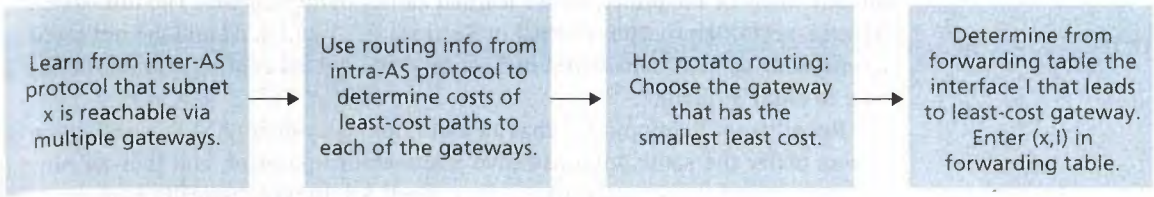


Figure 4.33 ♦ Steps in adding an outside-AS destination in a router's forwarding table

suppose AS1 learns from AS2 that subnet x is reachable via AS2. AS1 could then tell AS3 that x is reachable via AS1. In this manner, if AS3 needs to route a packet destined to x, AS3 would forward the packet to AS1, which would in turn forward the packet to AS2. As we'll see in our discussion of BGP, an AS has quite a bit of flexibility in deciding which destinations it advertises to its neighboring ASs. This is a *policy* decision, typically depending more on economic issues than on technical issues.

Recall from Section 1.5 that the Internet consists of a hierarchy of interconnected ISPs. So what is the relationship between ISPs and ASs? You might think that the routers in an ISP, and the links that interconnect them, constitute a single AS. Although this is often the case, many ISPs partition their network into multiple ASs. For example, some tier-1 ISPs use one AS for their entire network; others break up their ISP into tens of interconnected ASs.

In summary, the problems of scale and administrative authority are solved by defining autonomous systems. Within an AS, all routers run the same intra-AS routing protocol. Among themselves, the ASs run the same inter-AS routing protocol. The problem of scale is solved because an intra-AS router need only know about routers within its AS. The problem of administrative authority is solved since an organization can run whatever intra-AS routing protocol it chooses; however, each pair of connected ASs needs to run the same inter-AS routing protocol to exchange reachability information.

In the following section, we'll examine two intra-AS routing protocols (RIP and OSPF) and the inter-AS routing protocol (BGP) that are used in today's Internet. These case studies will nicely round out our study of hierarchical routing.

4.6 Routing in the Internet

Having studied Internet addressing and the IP protocol, we now turn our attention to the Internet's routing protocols; their job is to determine the path taken by a datagram between source and destination. We'll see that the Internet's routing protocols

DELL - Exhibit 1008 - Page 170

embody many of the principles we learned earlier in this chapter. The link-state and distance-vector approaches studied in Sections 4.5.1 and 4.5.2 and the notion of an autonomous system considered in Section 4.5.3 are all central to how routing is done in today's Internet.

Recall from Section 4.5.3 that an autonomous system (AS) is a collection of routers under the same administrative and technical control, and that all run the same routing protocol among themselves. Each AS, in turn, typically contains multiple subnets (where we use the term subnet in the precise, addressing sense in Section 4.4.2).

4.6.1 Intra-AS Routing in the Internet: RIP

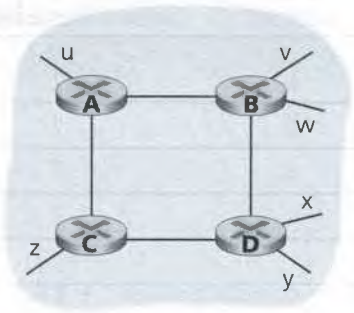
An intra-AS routing protocol is used to determine how routing is performed within an autonomous system (AS). Intra-AS routing protocols are also known as **interior gateway protocols**. Historically, two routing protocols have been used extensively for routing within an autonomous system in the Internet: the **Routing Information Protocol (RIP)** and **Open Shortest Path First (OSPF)**. A routing protocol closely related to OSPF is the **IS-IS** protocol [RFC 1142, Perlman 1999]. We first discuss RIP and then consider OSPF.

RIP was one of the earliest intra-AS Internet routing protocols and is still in widespread use today. It traces its origins and its name to the Xerox Network Systems (XNS) architecture. The widespread deployment of RIP was due in great part to its inclusion in 1982 in the Berkeley Software Distribution (BSD) version of UNIX supporting TCP/IP. RIP version 1 is defined in [RFC 1058], with a backward-compatible version 2 defined in [RFC 2453].

RIP is a distance-vector protocol that operates in a manner very close to the idealized DV protocol we examined in Section 4.5.2. The version of RIP specified in RFC 1058 uses hop count as a cost metric; that is, each link has a cost of 1. In the DV algorithm in Section 4.5.2, for simplicity, costs were defined between pairs of routers. In RIP (and also in OSPF), costs are actually from source router to a destination subnet. RIP uses the term *hop*, which is the number of subnets traversed along the shortest path from source router to destination subnet, including the destination subnet. Figure 4.34 illustrates an AS with six leaf subnets. The table in the figure indicates the number of hops from the source A to each of the leaf subnets.

The maximum cost of a path is limited to 15, thus limiting the use of RIP to autonomous systems that are fewer than 15 hops in diameter. Recall that in DV protocols, neighboring routers exchange distance vectors with each other. The distance vector for any one router is the current estimate of the shortest path distances from that router to the subnets in the AS. In RIP, routing updates are exchanged between neighbors approximately every 30 seconds using a **RIP response message**. The response message sent by a router or host contains a list of up to 25 destination subnets within the AS, as well as the sender's distance to each of those subnets. Response messages are also known as **RIP advertisements**.

DELL - Exhibit 1008 - Page 171



Destination	Hops
u	1
v	2
w	2
x	3
y	3
z	2

Figure 4.34 ♦ Number of hops from source router A to various subnets

Let's take a look at a simple example of how RIP advertisements work. Consider the portion of an AS shown in Figure 4.35. In this figure, lines connecting the routers denote subnets. Only selected routers (*A, B, C,* and *D*) and subnets (*w, x, y,* and *z*) are labeled. Dotted lines indicate that the AS continues on; thus this autonomous system has many more routers and links than are shown.

Each router maintains a RIP table known as a **routing table**. A router's routing table includes both the router's distance vector and the router's forwarding table. Figure 4.36 shows the routing table for router *D*. Note that the routing table has three columns. The first column is for the destination subnet, the second column indicates the identity of the next router along the shortest path to the destination subnet, and the third column indicates the number of hops (that is, the number of subnets that have to be traversed, including the destination subnet) to get to the destination subnet along the shortest path. For this example, the table indicates that to send a datagram from router *D* to destination subnet *w*, the datagram should first be forwarded to neighboring router *A;* the table also indicates that destination subnet *w* is two hops away along the shortest path. Similarly, the table indicates that subnet

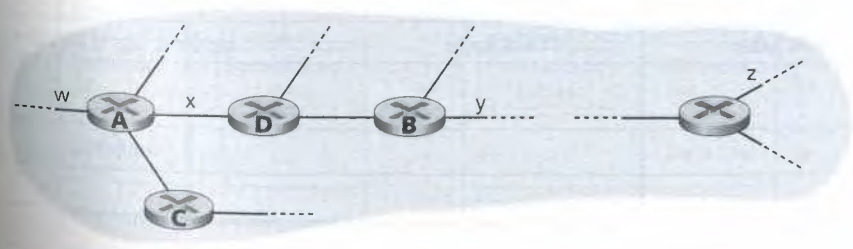


Figure 4.35 ♦ A portion of an autonomous system

DELL - Exhibit 1008 - Page 172

Destination Subnet	Next Router	Number of Hops to Destination
w	A	2
y	B	2
z	B	7
x	—	1
....	....	....

Figure 4.36 ♦ Routing table in router *D* before receiving advertisement from router *A*

z is seven hops away via router *B*. In principle, a routing table will have one row for each subnet in the AS, although RIP version 2 allows subnet entries to be aggregated using route aggregation techniques similar to those we examined in Section 4.4. The table in Figure 4.36, and the subsequent tables to come, are only partially complete.

Now suppose that 30 seconds later, router *D* receives from router *A* the advertisement shown in Figure 4.37. Note that this advertisement is nothing other than the routing table information from router *A*! This information indicates, in particular, that subnet *z* is only four hops away from router *A*. Router *D*, upon receiving this advertisement, merges the advertisement (Figure 4.37) with the old routing table (Figure 4.36). In particular, router *D* learns that there is now a path through router *A* to subnet *z* that is shorter than the path through router *B*. Thus, router *D* updates its routing table to account for the shorter shortest path, as shown in Figure 4.38. How is it, you might ask, that the shortest path to subnet *z* has become shorter? Possibly, the decentralized distance-vector algorithm is still in the process of converging (see Section 4.5.2), or perhaps new links and/or routers were added to the AS, thus changing the shortest paths in the AS.

Let's next consider a few of the implementation aspects of RIP. Recall that RIP routers exchange advertisements approximately every 30 seconds. If a router does

Destination Subnet	Next Router	Number of Hops to Destination
z	C	4
w	—	1
x	—	1
....	....	....

Figure 4.37 ♦ Advertisement from router A

DELL - Exhibit 1008 - Page 173

Destination Subnet	Next Router	Number of Hops to Destination
w	A	2
y	B	2
z	A	5
....	....	....

Figure 4.38 ♦ Routing table in router *D* after receiving advertisement from router *A*

not hear from its neighbor at least once every 180 seconds, that neighbor is considered to be no longer reachable; that is, either the neighbor has died or the connecting link has gone down. When this happens, RIP modifies the local routing table and then propagates this information by sending advertisements to its neighboring routers (the ones that are still reachable). A router can also request information about its neighbor's cost to a given destination using RIP's request message. Routers send RIP request and response messages to each other over UDP using port number 520. The UDP segment is carried between routers in a standard IP datagram. The fact that RIP uses a transport-layer protocol (UDP) on top of a network-layer protocol (IP) to implement network-layer functionality (a routing algorithm) may seem rather convoluted (it is!). Looking a little deeper at how RIP is implemented will clear this up.

Figure 4.39 sketches how RIP is typically implemented in a UNIX system, for example, a UNIX workstation serving as a router. A process called *routed* (pronounced "route dee") executes RIP, that is, maintains routing information and exchanges messages with *routed* processes running in neighboring routers. Because RIP is

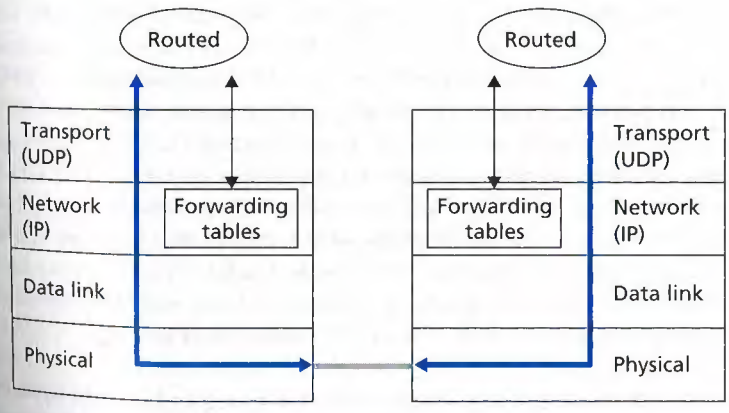


Figure 4.39 ♦ Implementation of RIP as the *routed* daemon

DELL - Exhibit 1008 - Page 174

implemented as an application-layer process (albeit a very special one that is able to manipulate the routing tables within the UNIX kernel), it can send and receive messages over a standard socket and use a standard transport protocol. As shown, RIP is implemented as an application-layer protocol (see Chapter 2) running over UDP.

4.6.2 Intra-AS Routing in the Internet: OSPF

Like RIP, OSPF routing is widely used for intra-AS routing in the Internet. OSPF and its closely related cousin, IS-IS, are typically deployed in upper-tier ISPs whereas RIP is deployed in lower-tier ISPs and enterprise networks. The Open in OSPF indicates that the routing protocol specification is publicly available (for example, as opposed to Cisco's EIGRP protocol). The most recent version of OSPF, version 2, is defined in RFC 2328, a public document.

OSPF was conceived as the successor to RIP and as such has a number of advanced features. At its heart, however, OSPF is a link-state protocol that uses flooding of link-state information and a Dijkstra least-cost path algorithm. With OSPF, a router constructs a complete topological map (that is, a graph) of the entire autonomous system. The router then locally runs Dijkstra's shortest-path algorithm to determine a shortest-path tree to all *subnets*, with itself as the root node. Individual link costs are configured by the network administrator (see Principles and Practice: Setting OSPF Weights). The administrator might choose to set all link costs to 1, thus achieving minimum-hop routing, or might choose to set the link weights to be inversely proportional to link capacity in order to discourage traffic from using low-bandwidth links. OSPF does not mandate a policy for how link weights are set (that is the job of the network administrator), but instead provides the mechanisms (protocol) for determining least-cost path routing for the given set of link weights.

With OSPF, a router broadcasts routing information to *all* other routers in the autonomous system, not just to its neighboring routers. A router broadcasts link-state information whenever there is a change in a link's state (for example, a change in cost or a change in up/down status). It also broadcasts a link's state periodically (at least once every 30 minutes), even if the link's state has not changed. RFC 2328 notes that "this periodic updating of link state advertisements adds robustness to the link state algorithm." OSPF advertisements are contained in OSPF messages that are carried directly by IP, with an upper-layer protocol of 89 for OSPF. Thus, the OSPF protocol must itself implement functionality such as reliable message transfer and link-state broadcast. The OSPF protocol also checks that links are operational (via a HELLO message that is sent to an attached neighbor) and allows an OSPF router to obtain a neighboring router's database of network-wide link state.

Some of the advances embodied in OSPF include the following:

- *Security.* Exchanges between OSPF routers (for example, link-state updates) can be authenticated. With authentication, only trusted routers can participate in the OSPF protocol within an AS, thus preventing malicious intruders (or networking

DELL - Exhibit 1008 - Page 175

students taking their newfound knowledge out for a joyride) from injecting incorrect information into router tables. By default, OSPF packets between routers are not authenticated and could be forged. Two types of authentication can be configured—simple and MD5 (see Chapter 8 for a discussion on MD5 and authentication in general). With simple authentication, the same password is configured on each router. When a router sends an OSPF packet, it includes the password in plaintext. Clearly, simple authentication is not very secure. MD5 authentication is based on shared secret keys that are configured in all the routers. For each OSPF packet that it sends, the router computes the MD5 hash of the content of the OSPF packet appended with the secret key. (See the discussion of message authentication codes in Chapter 7.) Then the router includes the resulting hash value in the OSPF packet. The receiving router, using the preconfigured secret key, will compute an MD5 hash of the packet and compare it with the hash value that the packet carries, thus verifying the packet's authenticity. Sequence numbers are also used with MD5 authentication to protect against replay attacks.

- *Multiple same-cost paths.* When multiple paths to a destination have the same cost, OSPF allows multiple paths to be used (that is, a single path need not be chosen for carrying all traffic when multiple equal-cost paths exist).
- *Integrated support for unicast and multicast routing.* Multicast OSPF (MOSPF) [RFC 1584] provides simple extensions to OSPF to provide for multicast routing (a topic we cover in more depth in Section 4.7.2). MOSPF uses the existing OSPF link database and adds a new type of link-state advertisement to the existing OSPF link-state broadcast mechanism.
- *Support for hierarchy within a single routing domain.* Perhaps the most significant advance in OSPF is the ability to structure an autonomous system hierarchically. Section 4.5.3 has already looked at the many advantages of hierarchical routing structures. We cover the implementation of OSPF hierarchical routing in the remainder of this section.

An OSPF autonomous system can be configured into areas. Each area runs its own OSPF link-state routing algorithm, with each router in an area broadcasting its link state to all other routers in that area. The internal details of an area thus remain invisible to all routers outside the area. Intra-area routing involves only those routers within the same area.

Within each area, one or more **area border routers** are responsible for routing packets outside the area. Exactly one OSPF area in the AS is configured to be the **backbone** area. The primary role of the backbone area is to route traffic between the other areas in the AS. The backbone always contains all area border routers in the AS and may contain nonborder routers as well. Inter-area routing within the AS requires that the packet be first routed to an area border router (intra-area routing), then routed through the backbone to the area border router that is in the destination area, and then routed to the final destination.

DELL - Exhibit 1008 - Page 176

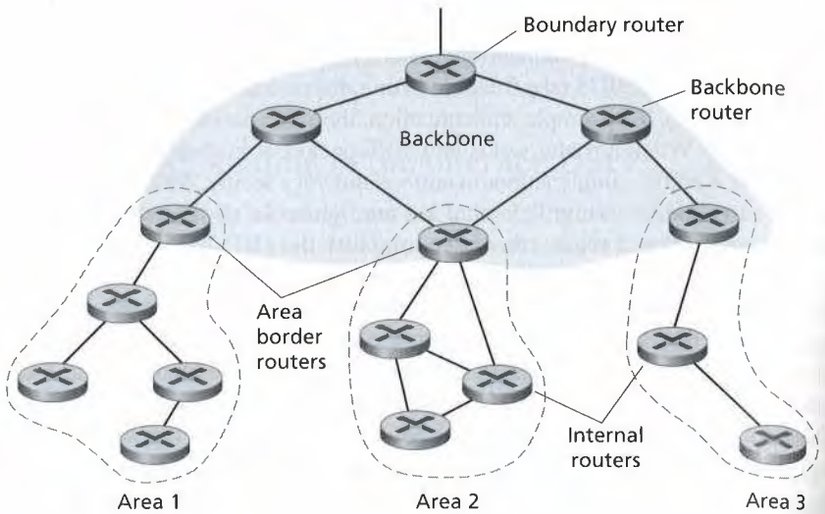


Figure 4.40 ♦ Hierarchically structured OSPF AS with four areas

A diagram of a hierarchically structured OSPF network is shown in Figure 4.40. We can identify four types of OSPF routers in Figure 4.40:

- *Internal routers.* These routers are in nonbackbone areas and perform only intra-AS routing.
- *Area border routers.* These routers belong to both an area and the backbone.
- *Backbone routers (nonborder routers).* These routers perform routing within the backbone but themselves are not area border routers. Within a nonbackbone area, internal routers learn of the existence of routes to other areas from information (essentially a link-state advertisement, but advertising the cost of a route to another area, rather than a link cost) broadcast within the area by its backbone routers.
- *Boundary routers.* A boundary router exchanges routing information with routers belonging to other autonomous systems. This router might, for example, use BGP to perform inter-AS routing. It is through such a boundary router that other routers learn about paths to external networks.

OSPF is a relatively complex protocol, and our coverage here has been necessarily brief; [Huitema 1998; Moy 1998; RFC 2328] provide additional details.

DELL - Exhibit 1008 - Page 177

PRINCIPLES IN PRACTICE

SETTING OSPF LINK WEIGHTS

Our discussion of link-state routing has implicitly assumed that link weights are set, a routing algorithm such as OSPF is run, and traffic flows according to the routing tables computed by the LS algorithm. In terms of cause and effect, the link weights are given (i.e., they come first) and result (via Dijkstra's algorithm) in routing paths that minimize overall cost. In this viewpoint, link weights reflect the cost of using a link (e.g., if link weights are inversely proportional to capacity, then the use of high-capacity links would have smaller weights and thus be more attractive from a routing standpoint) and Disjkstra's algorithm serves to minimize overall cost.

In practice, the cause and effect relationship between link weights and routing paths may be reversed, with network operators configuring link weights in order to obtain routing paths that achieve certain traffic engineering goals [Fortz 2000, Fortz 2002]. For example, suppose a network operator has an estimate of traffic flow entering the network at each ingress point and destined for each egress point. The operator may then want to put in place a specific routing of ingress-to-egress flows that minimizes the maximum utilization over all of the network's links. But with a routing algorithm such as OSPF, the operator's main "knobs" for tuning the routing of flows through the network are the link weights. Thus, in order to achieve the goal of minimizing the maximum link utilization, the operator must find the set of link weights that achieves this goal. This is a reversal of the cause and effect relationship—the desired routing of flows is known, and the OSPF link weights must be found such that the OSPF routing algorithm results in this desired routing of flows.

4.6.3 Inter–AS Routing: BGP

We just learned how ISPs use RIP and OSPF to determine optimal paths for source-destination pairs that are internal to the same AS. Let's now examine how paths are determined for source-destination pairs that span multiple ASs. The **Border Gateway Protocol** version 4, specified in RFC 4271 (see also [RFC 1772; RFC 1773]), is the *de facto* standard inter-AS routing protocol in today's Internet. It is commonly referred to as BGP4 or simply as **BGP**. As an inter-AS routing protocol (see Section 4.5.3), BGP provides each AS a means to

1. Obtain subnet reachability information from neighboring ASs.
2. Propagate the reachability information to all routers internal to the AS.
3. Determine "good" routes to subnets based on the reachability information and on AS policy.

DELL - Exhibit 1008 - Page 178

Most importantly, BGP allows each subnet to advertise its existence to the rest of the Internet. A subnet screams "I exist and I am here," and BGP makes sure that all the ASs in the Internet know about the subnet and how to get there. If it weren't for BGP, each subnet would be isolated—alone and unknown by the rest of the Internet.

BGP Basics

BGP is extremely complex; entire books have been devoted to the subject and many issues are still not well understood [Yannuzzi 2005]. Furthermore, even after having read the books and RFCs, you may find it difficult to fully master BGP without having practiced BGP for many months (if not years) as a designer or administrator of an upper-tier ISP. Nevertheless, because BGP is an absolutely critical protocol for the Internet—in essence, it is the protocol that glues the whole thing together—we need to acquire at least a rudimentary understanding of how it works. We begin by describing how BGP might work in the context of the simple example network we studied earlier in Figure 4.32. In this description, we build on our discussion of hierarchical routing in Section 4.5.3; we encourage you to review that material.

In BGP, pairs of routers exchange routing information over semipermanent TCP connections using port 179. The semi-permanent TCP connections for the network in Figure 4.32 are shown in Figure 4.41. There is typically one such BGP TCP connection for each link that directly connects two routers in two different ASs; thus, in Figure 4.41, there is a TCP connection between gateway routers 3a and 1c and another TCP connection between gateway routers 1b and 2a. There are also semipermanent BGP TCP connections between routers within an AS. In particular, Figure 4.41 displays a common configuration of one TCP connection for each pair of routers internal to an AS, creating a mesh of TCP connections within each AS. For each TCP connection, the two routers at the end of the connection are called **BGP peers**, and the TCP connection along with all the BGP messages sent over the

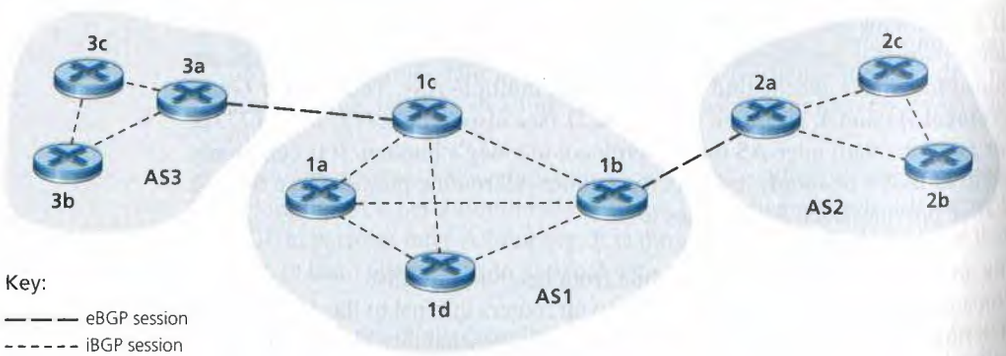


Figure 4.41 ♦ eBGP and iBGP sessions

DELL - Exhibit 1008 - Page 179

connection is called a **BGP session**. Furthermore, a BGP session that spans two ASs is called an **external BGP (eBGP) session**, and a BGP session between routers in the same AS is called an **internal BGP (iBGP) session**. In Figure 4.41, the eBGP sessions are shown with the long dashes; the iBGP sessions are shown with the short dashes. Note that BGP session lines in Figure 4.41 do not always correspond to the physical links in Figure 4.32.

BGP allows each AS to learn which destinations are reachable via its neighboring ASs. In BGP, destinations are not hosts but instead are CIDRized **prefixes**, with each prefix representing a subnet or a collection of subnets. Thus, for example, suppose there are four subnets attached to AS2: 138.16.64/24, 138.16.65/24, 138.16.66/24, and 138.16.67/24. Then AS2 could aggregate the prefixes for these four subnets and use BGP to advertise the single prefix to 138.16.64/22 to AS1. As another example, suppose that only the first three of those four subnets are in AS2 and the fourth subnet, 138.16.67/24, is in AS3. Then, as described in the Principles and Practice in Section 4.4.2, because routers use longest-prefix matching for forwarding datagrams, AS3 could advertise to AS1 the more specific prefix 138.16.67/24 and AS2 could *still* advertise to AS1 the aggregated prefix 138.16.64/22.

Let's now examine how BGP would distribute prefix reachability information over the BGP sessions shown in Figure 4.41. As you might expect, using the eBGP session between the gateway routers 3a and 1c, AS3 sends AS1 the list of prefixes that are reachable from AS3; and AS1 sends AS3 the list of prefixes that are reachable from AS1. Similarly, AS1 and AS2 exchange prefix reachability information through their gateway routers 1b and 2a. Also as you may expect, when a gateway router (in any AS) receives eBGP-learned prefixes, the gateway router uses its iBGP sessions to distribute the prefixes to the other routers in the AS. Thus, all the routers in AS1 learn about AS3 prefixes, including the gateway router 1b. The gateway router 1b (in AS1) can therefore re-advertise AS3's prefixes to AS2. When a router (gateway or not) learns about a new prefix, it creates an entry for the prefix in its forwarding table, as described in Section 4.5.3.

Path Attributes and BGP Routes

Having now a preliminary understanding of BGP, let's get a little deeper into it (while still brushing some of the less important details under the rug!). In BGP, an autonomous system is identified by its globally unique **autonomous system number (ASN)** [RFC 1930]. (Technically, not every AS has an ASN. In particular, a so-called stub AS that carries only traffic for which it is a source or destination will not typically have an ASN; we ignore this technicality in our discussion in order to better see the forest for the trees.) AS numbers, like IP addresses, are assigned by ICANN regional registries [ICANN 2007].

When a router advertises a prefix across a BGP session, it includes with the prefix a number of **BGP attributes**. In BGP jargon, a prefix along with its attributes is

DELL - Exhibit 1008 - Page 180

called a **route**. Thus, BGP peers advertise routes to each other. Two of the more important attributes are AS-PATH and NEXT-HOP:

- *AS-PATH*. This attribute contains the ASs through which the advertisement for the prefix has passed. When a prefix is passed into an AS, the AS adds its ASN to the AS-PATH attribute. For example, consider Figure 4.41 and suppose that prefix 138.16.64/24 is first advertised from AS2 to AS1; if AS1 then advertises the prefix to AS3, AS-PATH would be AS2 AS1. Routers use the AS-PATH attribute to detect and prevent looping advertisements; specifically, if a router sees that its AS is contained in the path list, it will reject the advertisement. As we'll soon discuss, routers also use the AS-PATH attribute in choosing among multiple paths to the same prefix.
- Providing the critical link between the inter-AS and intra-AS routing protocols, the NEXT-HOP attribute has a subtle but important use. *The NEXT-HOP is the router interface that begins the AS-PATH*. To gain insight into this attribute, let's again refer to Figure 4.41. Consider what happens when the gateway router 3a in AS3 advertises a route to gateway router 1c in AS1 using eBGP. The route includes the advertised prefix, which we'll call x, and an AS-PATH to the prefix. This advertisement also includes the NEXT-HOP, which is the IP address of the router 3a interface that leads to 1c. (Recall that a router has multiple IP addresses, one for each of its interfaces.) Now consider what happens when router 1d learns about this route from iBGP. After learning about this route to x, router 1d may want to forward packets to x along the route, that is, router 1d may want to include the entry (x, I) in its forwarding table, where I is its interface that begins the least-cost path from 1d towards the gateway router 1c. To determine I, 1d provides the IP address in the NEXT-HOP attribute to its intra-AS routing module. Note that the intra-AS routing algorithm has determined the least-cost path to all subnets attached to the routers in AS1, including to the subnet for the link between 1c and 3a. From this least-cost path from 1d to the 1c-3a subnet, 1d determines its router interface I that begins this path and then adds the entry (x, I) to its forwarding table. Whew! In summary, the AS-PATH attribute is used by routers to properly configure their forwarding tables.

Figure 4.42 illustrates another situation where the AS-PATH is needed. In this figure, AS1 and AS2 are connected by two peering links. A router in AS1 could learn about two different routes to the same prefix x. These two routes could have the same AS-PATH to x, but could have different NEXT-HOP values corresponding to the different peering links. Using the AS-PATH values and the intra-AS routing algorithm, the router can determine the cost of the path to each peering link, and then apply hot-potato routing (see Section 4.5.3) to determine the appropriate interface.

BGP also includes attributes that allow routers to assign preference metrics to the routes, and an attribute that indicates how the prefix was inserted into BGP at

DELL - Exhibit 1008 - Page 181

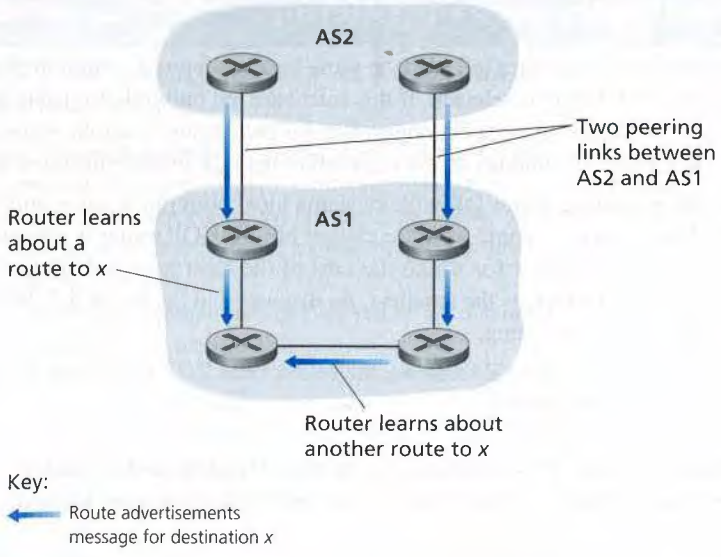


Figure 4.42 ♦ NEXT-HOP attributes in advertisements are used to determine which peering link to use.

the origin AS. For a full discussion of route attributes, see [Griffin 2002; Stewart 1999; Halabi 2000; Feamster 2004; RFC 4271].

When a gateway router receives a router advertisement, it uses its **import policy** to decide whether to accept or filter the route and whether to set certain attributes such as the router preference metrics. The import policy may filter a route because the AS may not want to send traffic over one of the ASs in the route's AS-PATH. The gateway router may also filter a route because it already knows of a preferable route to the same prefix.

BGP Route Selection

As described earlier in this section, BGP uses eBGP and iBGP to distribute routes to all the routers within ASs. From this distribution, a router may learn about more than one route to any one prefix, in which case the router must select one of the possible routes. The inputs into this route selection process is the set of all routes that have been learned and accepted by the router. If there are two or more routes to the same prefix, then BGP sequentially invokes the following elimination rules until one route remains:

- Routes are assigned a local preference value as one of their attributes. The local preference of a route could have been set by the router or could have been learned by another router in the same AS. This is a policy decision that is left up

DELL - Exhibit 1008 - Page 182

to the AS's network administrator. (We will shortly discuss BGP policy issues in some detail.) The routes with the highest local preference values are selected.

- From the remaining routes (all with the same local preference value), the route with the shortest AS-PATH is selected. If this rule were the only rule for route selection, then BGP would be using a DV algorithm for path determination, where the distance metric uses the number of AS hops rather than the number of router hops.
- From the remaining routes (all with the same local preference value and the same AS-PATH length), the route with the closest NEXT-HOP router is selected. Here, closest means the router for which the cost of the least-cost path, determined by the intra-AS algorithm, is the smallest. As discussed in Section 4.5.3, this process is called hot-potato routing.
- If more than one route still remains, the router uses BGP identifiers to select the route; see [Stewart 1999].

The elimination rules are even more complicated than described above. To avoid nightmares about BGP, it's best to learn about BGP selection rules in small doses!

Routing Policy

Let's illustrate some of the basic concepts of BGP routing policy with a simple example. Figure 4.43 shows six interconnected autonomous systems: A, B, C, W, X, and Y. It is important to note that A, B, C, W, X, and Y are ASs, not routers. Let's assume that autonomous systems W, X, and Y are stub networks and that A, B, and C are backbone provider networks. We'll also assume that A, B, and C, all peer with each other, and provide full BGP information to their customer networks. All traffic entering a **stub network** must be destined for that network, and all traffic leaving a stub network must have originated in that network. W and Y are clearly stub networks. X is a **multi-homed stub network,** since it is connected to the rest of the network via two different providers (a scenario that is becoming increasingly common in practice). However, like W and Y, X itself must be the source/destination of all traffic leaving/entering X. But how will this stub network behavior be implemented and enforced? How will X be prevented from forwarding traffic between B and C? This can easily be

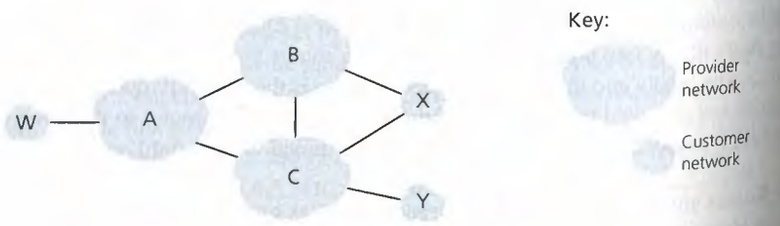


Figure 4.43 ♦ A simple BGP scenario

DELL - Exhibit 1008 - Page 183

PRINCIPLES IN PRACTICE

WHY ARE THERE DIFFERENT INTER-AS AND INTRA-AS ROUTING PROTOCOLS?

Having now studied the details of specific inter-AS and intra-AS routing protocols deployed in today's Internet, let's conclude by considering perhaps the most fundamental question we could ask about these protocols in the first place (hopefully, you have been wondering this all along, and have not lost the forest for the trees!): Why are different inter-AS and intra-AS routing protocols used?

The answer to this question gets at the heart of the differences between the goals of routing within an AS and among ASs:

- *Policy.* Among ASs, policy issues dominate. It may well be important that traffic originating in a given AS not be able to pass through another specific AS. Similarly, a given AS may well want to control what transit traffic it carries between other ASs. We have seen that BGP carries path attributes and provides for controlled distribution of routing information so that such policy-based routing decisions can be made. Within an AS, everything is nominally under the same administrative control, and thus policy issues play a much less important role in choosing routes within the AS.
- *Scale.* The ability of a routing algorithm and its data structures to scale to handle routing to/among large numbers of networks is a critical issue in inter-AS routing. Within an AS, scalability is less of a concern. For one thing, if a single administrative domain becomes too large, it is always possible to divide it into two ASs and perform inter-AS routing between the two new ASs. (Recall that OSPF allows such a hierarchy to be built by splitting an AS into areas.)
- *Performance.* Because inter-AS routing is so policy oriented, the quality (for example, performance) of the routes used is often of secondary concern (that is, a longer or more costly route that satisfies certain policy criteria may well be taken over a route that is shorter but does not meet that criteria). Indeed, we saw that among ASs, there is not even the notion of cost (other than AS hop count) associated with routes. Within a single AS, however, such policy concerns are of less importance, allowing routing to focus more on the level of performance realized on a route.

accomplished by controlling the manner in which BGP routes are advertised. In particular, X will function as a stub network if it advertises (to its neighbors B and C) that it has no paths to any other destinations except itself. That is, even though X may know of a path, say XCY, that reaches network Y, it will *not* advertise this path to B. Since B is unaware that X has a path to Y, B would never forward traffic destined to Y (or C) via X. This simple example illustrates how a selective route advertisement policy can be used to implement customer/provider routing relationships.

DELL - Exhibit 1008 - Page 184

Let's next focus on a provider network, say AS B. Suppose that B has learned (from A) that A has a path AW to W. B can thus install the route BAW into its routing information base. Clearly, B also wants to advertise the path BAW to its customer, X, so that X knows that it can route to W via B. But should B advertise the path BAW to C? If it does so, then C could route traffic to W via CBAW. If A, B, and C are all backbone providers, than B might rightly feel that it should not have to shoulder the burden (and cost!) of carrying transit traffic between A and C. B might rightly feel that it is A's and C's job (and cost!) to make sure that C can route to/from A's customers via a direct connection between A and C. There are currently no official standards that govern how backbone ISPs route among themselves. However, a rule of thumb followed by commercial ISPs is that any traffic flowing across an ISP's backbone network must have either a source or a destination (or both) in a network that is a customer of that ISP; otherwise the traffic would be getting a free ride on the ISP's network. Individual peering agreements (that would govern questions such as those raised above) are typically negotiated between pairs of ISPs and are often confidential; [Huston 1999a] provides an interesting discussion of peering agreements. For a detailed description of how routing policy reflects commercial relationships among ISPs, see [Gao 2001]. For a recent discussion of BGP routing polices from an ISP standpoint, see [Caesar 2005].

As noted above, BGP is the *de facto* standard for inter-AS routing for the public Internet. To see the contents of various BGP routing tables (large!) extracted from routers in tier-1 ISPs, see http://www.routeviews.org. BGP routing tables often contain tens of thousands of prefixes and corresponding attributes. Statistics about the size and characteristics of BGP routing tables are presented in [Huston 2001; Meng 2005].

This completes our brief introduction to BGP. Understanding BGP is important because it plays a central role in the Internet. We encourage you to see the references [Griffin 2002; Stewart 1999; Labovitz 1997; Halabi 2000; Huitema 1998; Gao 2001; Feamster 2004; Caesar 2005] to learn more about BGP.

4.7 Broadcast and Multicast Routing

Thus far in this chapter, our focus has been on routing protocols that support unicast (i.e., point-to-point) communication, in which a single source node sends a packet to a single destination node. In this section, we turn our attention to broadcast and multicast routing protocols. In **broadcast routing**, the network layer provides a service of delivering a packet sent from a source node to all other nodes in the network; **multicast routing** enables a single source node to send a copy of a packet to a subset of the other network nodes. In Section 4.7.1 we'll consider broadcast routing algorithms and their embodiment in routing protocols. We'll examine multicast routing in Section 4.7.2.

DELL - Exhibit 1008 - Page 185

4.7.1 Broadcast Routing Algorithms

Perhaps the most straightforward way to accomplish broadcast communication is for the sending node to send a separate copy of the packet to each destination, as shown in Figure 4.44(a). Given N destination nodes, the source node simply makes N copies of the packet, addresses each copy to a different destination, and then transmits the N copies to the N destinations using unicast routing. This **N-way-unicast** approach to broadcasting is simple—no new network-layer routing protocol, packet-duplication, or forwarding functionality is needed. There are, however, several drawbacks to this approach. The first drawback is its inefficiency. If the source node is connected to the rest of the network via a single link, then N separate copies of the (same) packet will traverse this single link. It would clearly be more efficient to send only a single copy of a packet over this first hop and then have the node at the other end of the first hop make and forward any additional needed copies. That is, it would be more efficient for the network nodes themselves (rather than just the source node) to create duplicate copies of a packet. For example, in Figure 4.44(b), only a single copy of a packet traverses the R1-R2 link. That packet is then duplicated at R2, with a single copy being sent over links R2-R3 and R2-R4.

The additional drawbacks of N-way-unicast are perhaps more subtle, but no less important. An implicit assumption of N-way-unicast is that broadcast recipients, and their addresses, are known to the sender. But how is this information obtained? Most likely, additional protocol mechanisms (such as a broadcast membership or destination-registration protocol) would be required. This would add more overhead and, importantly, additional complexity to a protocol that had initially seemed quite simple. A final drawback of N-way-unicast relates to the purposes for which broadcast is to be used. In Section 4.5, we learned that link-state routing protocols use broadcast to disseminate the link-state information that is used to compute unicast routes. Clearly, in situations where broadcast is used to create and update unicast routes, it would be unwise (at best!) to rely on the unicast routing infrastructure to achieve broadcast.

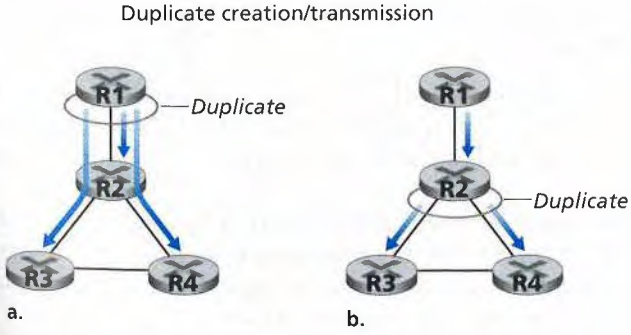


Figure 4.44 ♦ Source-duplication versus in-network duplication

DELL - Exhibit 1008 - Page 186

Given the several drawbacks of N-way-unicast broadcast, approaches in which the network nodes themselves play an active role in packet duplication, packet forwarding, and computation of the broadcast routes are clearly of interest. We'll examine several such approaches below and again adopt the graph notation introduced in Section 4.5. We again model the network as a graph, $G = (N,E)$, where N is a set of nodes and a collection E of edges, where each edge is a pair of nodes from N. We'll be a bit sloppy with our notation and use N to refer to both the set of nodes, as well as the cardinality ($|N|$) or size of that set when there is no confusion.

Uncontrolled Flooding

The most obvious technique for achieving broadcast is a **flooding** approach in which the source node sends a copy of the packet to all of its neighbors. When a node receives a broadcast packet, it duplicates the packet and forwards it to all of its neighbors (except the neighbor from which it received the packet). Clearly, if the graph is connected, this scheme will eventually deliver a copy of the broadcast packet to all nodes in the graph. Although this scheme is simple and elegant, it has a fatal flaw (before you read on, see if you can figure out this fatal flaw): If the graph has cycles, then one or more copies of each broadcast packet will cycle indefinitely. For example, in Figure 4.44, R2 will flood to R3, R3 will flood to R4, R4 will flood to R2, and R2 will flood (again!) to R3, and so on. This simple scenario results in the endless cycling of two broadcast packets, one clockwise, and one counterclockwise. But there can be an even more calamitous fatal flaw: When a node is connected to more than two other nodes, it will create and forward multiple copies of the broadcast packet, each of which will create multiple copies of itself (at other nodes with more than two neighbors), and so on. This **broadcast storm**, resulting from the endless multiplication of broadcast packets, would eventually result in so many broadcast packets being created that the network would be rendered useless. (See the homework questions at the end of the chapter for a problem analyzing the rate at which such a broadcast storm grows.)

Controlled Flooding

The key to avoiding a broadcast storm is for a node to judiciously choose when to flood a packet and (e.g., if it has already received and flooded an earlier copy of a packet) when not to flood a packet. In practice, this can be done in one of several ways.

In **sequence-number-controlled flooding**, a source node puts its address (or other unique identifier) as well as a **broadcast sequence number** into a broadcast packet, then sends the packet to all of its neighbors. Each node maintains a list of the source address and sequence number of each broadcast packet it has already received, duplicated, and forwarded. When a node receives a broadcast packet, it first checks whether the packet is in this list. If so, the packet is dropped; if not, the

DELL - Exhibit 1008 - Page 187

packet is duplicated and forwarded to all the node's neighbors (except the node from which the packet has just been received). The Gnutella protocol, discussed in Chapter 2, uses sequence-number-controlled flooding to broadcast queries in its overlay network. (In Gnutella, message duplication and forwarding is performed at the application layer rather than at the network layer.)

A second approach to controlled flooding is known as **reverse path forwarding (RPF)** [Dalal 1978], also sometimes referred to as reverse path broadcast (RPB). The idea behind RPF is simple, yet elegant. When a router receives a broadcast packet with a given source address, it transmits the packet on all of its outgoing links (except the one on which it was received) only if the packet arrived on the link that is on its own shortest unicast path back to the source. Otherwise, the router simply discards the incoming packet without forwarding it on any of its outgoing links. Such a packet can be dropped because the router knows it either will receive or has already received a copy of this packet on the link that is on its own shortest path back to the sender. (You might want to convince yourself that this will, in fact, happen and that looping and broadcast storms will not occur.) Note that RPF does not use unicast routing to actually deliver a packet to a destination, nor does it require that a router know the complete shortest path from itself to the source. RPF need only know the next neighbor on its unicast shortest path to the sender; it uses this neighbor's identity only to determine whether or not to flood a received broadcast packet.

Figure 4.45 illustrates RPF. Suppose that the links drawn with thick lines represent the least-cost paths from the receivers to the source (*A*). Node *A* initially broadcasts a source-*A* packet to nodes *C* and *B*. Node *B* will forward the source-*A* packet it has received from *A* (since *A* is on its least-cost path to *A*) to both *C* and *D*. *B* will ignore (drop, without forwarding) any source-*A* packets it receives from any other

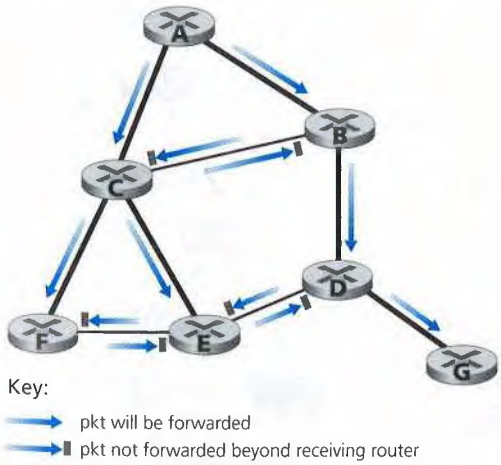


Figure 4.45 ♦ Reverse path forwarding

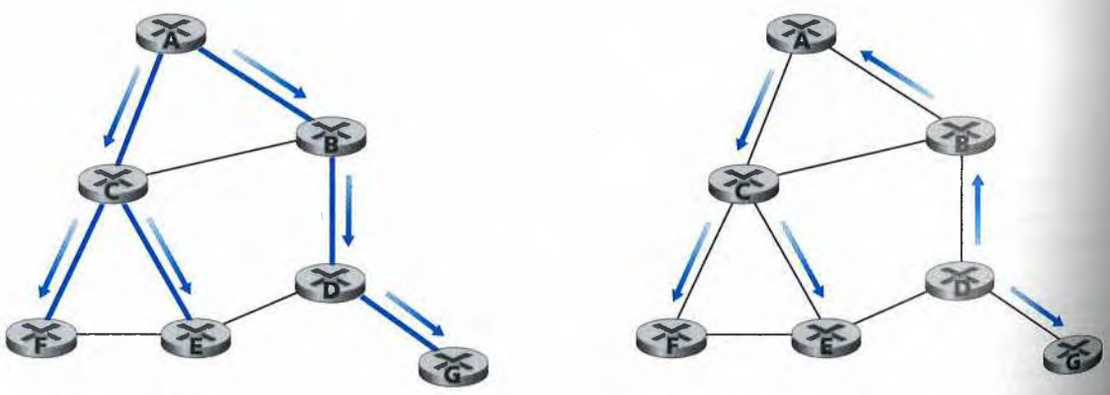
DELL - Exhibit 1008 - Page 188

nodes (for example, from routers C or D). Let us now consider node C, which will receive a source-A packet directly from A as well as from B. Since B is not on C's own shortest path back to A, C will ignore any source-A packets it receives from B. On the other hand, when C receives a source-A packet directly from A, it will forward the packet to nodes B, E, and F.

Spanning-Tree Broadcast

While sequence-number-controlled flooding and RPF avoid broadcast storms, they do not completely avoid the transmission of redundant broadcast packets. For example, in Figure 4.46, nodes B, C, D, E, and F receive either one or two redundant packets. Ideally, every node should receive only one copy of the broadcast packet. Examining the tree consisting of the nodes connected by thick lines in Figure 4.46(a), you can see that if broadcast packets were forwarded only along links within this tree, each and every network node would receive exactly one copy of the broadcast packet—exactly the solution we were looking for! This tree is an example of a **spanning tree**—a tree that contains each and every node in a graph. More formally, a spanning tree of a graph $G = (N,E)$ is a graph $G' = (N,E')$ such that E' is a subset of E, G' is connected, G' contains no cycles, and G' contains all the original nodes in G. If each link has an associated cost and the cost of a tree is the sum of the link costs, then a spanning tree whose cost is the minimum of all of the graph's spanning trees is called (not surprisingly) a **minimum spanning tree**.

Thus, another approach to providing broadcast is for the network nodes to first construct a spanning tree. When a source node wants to send a broadcast packet, it sends the packet out on all of the incident links that belong to the spanning tree. A node receiving a broadcast packet then forwards the packet to all its neighbors in the



a. Broadcast initiated at A

b. Broadcast initiated at D

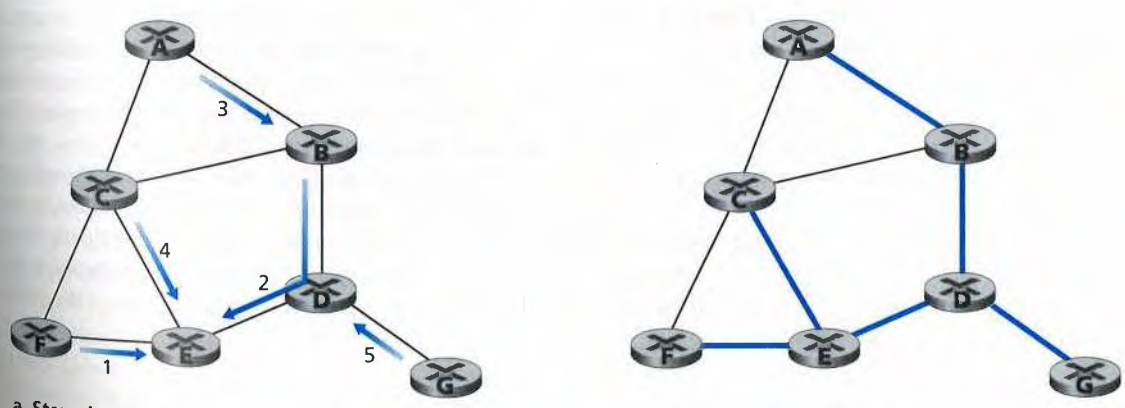
Figure 4.46 ♦ Broadcast along a spanning tree

DELL - Exhibit 1008 - Page 189

spanning tree (except the neighbor from which it received the packet). Not only does spanning tree eliminate redundant broadcast packets, but once in place, the spanning tree can be used by any node to begin a broadcast, as shown in Figures 4.46(a) and 4.46(b). Note that a node need not be aware of the entire tree; it simply needs to know which of its neighbors in G are spanning-tree neighbors.

The main complexity associated with the spanning-tree approach is the creation and maintenance of the spanning tree. Numerous distributed spanning-tree algorithms have been developed [Gallager 1983, Gartner 2003]. We consider only one simple algorithm here. In the **center-based approach** to building a spanning tree, a center node (also known as a **rendezvous point** or a **core**) is defined. Nodes then unicast tree-join messages addressed to the center node. A tree-join message is forwarded using unicast routing toward the center until it either arrives at a node that already belongs to the spanning tree or arrives at the center. In either case, the path that the tree-join message has followed defines the branch of the spanning tree between the edge node that initiated the tree-join message and the center. One can think of this new path as being grafted onto the existing spanning tree.

Figure 4.47 illustrates the construction of a center-based spanning tree. Suppose that node E is selected as the center of the tree. Suppose that node F first joins the tree and forwards a tree-join message to E. The single link EF becomes the initial spanning tree. Node B then joins the spanning tree by sending its tree-join message to E. Suppose that the unicast path route to E from B is via D. In this case, the tree-join message results in the path BDE being grafted onto the spanning tree. Node A next joins the spanning group by forwarding its tree-join message towards E. If A's unicast path to E is through B, then since B has already joined the spanning tree, the arrival of A's tree-join message at B will result in the AB link being immediately grafted onto the spanning tree. Node C joins the spanning tree next by forwarding



a. Stepwise construction of spanning tree

b. Constructed spanning tree

Figure 4.47 ♦ Center-based construction of a spanning tree

DELL - Exhibit 1008 - Page 190

its tree-join message directly to E. Finally, because the unicast routing from G to E must be via node D, when G sends its tree-join message to E, the GD link is grafted onto the spanning tree at node D.

Broadcast Algorithms in Practice

Broadcast protocols are used in practice at both the application and network layers. As discussed in Section 2.6, Gnutella [Gnutella 2007] uses application-level broadcast in order to broadcast queries for content among Gnutella peers. Here, a link between two distributed application-level peer processes in the Gnutella network is actually a TCP connection. Gnutella uses a form of sequence-number-controlled flooding in which a 16-bit identifier and a 16-bit payload descriptor (which identifies the Gnutella message type) are used to detect whether a received broadcast query has been previously received, duplicated, and forwarded. As also discussed in Section 2.6, Gnutella also uses a time-to-live (TTL) field to limit the number of hops over which a flooded query will be forwarded. When a Gnutella process receives and duplicates a query, it decrements the TTL field before forwarding the query. Thus, a flooded Gnutella query will only reach peers that are within a given number (the initial value of TTL) of application-level hops from the query initiator. Gnutella's flooding mechanism is thus sometimes referred to as *limited-scope flooding*.

A form of sequence-number-controlled flooding is also used to broadcast link-state advertisements (LSAs) in the OSPF [RFC 2328, Perlman 1999] routing algorithm, and in the Intermediate-System-to-Intermediate-System (IS-IS) routing algorithm [RFC 1142, Perlman 1999]. OSPF uses a 32-bit sequence number, as well as a 16-bit age field to identify LSAs. Recall that an OSPF node broadcasts LSAs for its attached links periodically, when a link cost to a neighbor changes, or when a link goes up/down. LSA sequence numbers are used to detect duplicate LSAs, but also serve a second important function in OSPF. With flooding, it is possible for an LSA generated by the source at time t to arrive *after* a newer LSA that was generated by the same source at time $t + \delta$. The sequence numbers used by the source node allow an older LSA to be distinguished from a newer LSA. The age field serves a purpose similar to that of a TTL value. The initial age field value is set to zero and is incremented at each hop as it flooded, and is also incremented as it sits in a router's memory waiting to be flooded. Although we have only briefly described the LSA flooding algorithm here, we note that designing LSA broadcast protocols can be very tricky business indeed. [RFC 789; Perlman 1999] describe an incident in which incorrectly transmitted LSAs by two malfunctioning routers caused an early version of an LSA flooding algorithm to take down the entire ARPAnet!

4.7.2 Multicast

We've seen in the previous section that with broadcast service, packets are delivered to each and every node in the network. In this section we turn our attention to

DELL - Exhibit 1008 - Page 191

multicast service, in which a multicast packet is delivered to only a *subset* of network nodes. A number of emerging network applications require the delivery of packets from one or more senders to a group of receivers. These applications include bulk data transfer (for example, the transfer of a software upgrade from the software developer to users needing the upgrade), streaming continuous media (for example, the transfer of the audio, video, and text of a live lecture to a set of distributed lecture participants), shared data applications (for example, a whiteboard or teleconferencing application that is shared among many distributed participants), data feeds (for example, stock quotes), Web cache updating, and interactive gaming (for example, distributed interactive virtual environments or multiplayer games).

In multicast communication, we are immediately faced with two problems—how to identify the receivers of a multicast packet and how to address a packet sent to these receivers. In the case of unicast communication, the IP address of the receiver (destination) is carried in each IP unicast datagram and identifies the single recipient; in the case of broadcast, *all* nodes need to receive the broadcast packet, so no destination addresses are needed. But in the case of multicast, we now have multiple receivers. Does it make sense for each multicast packet to carry the IP addresses of all of the multiple recipients? While this approach might be workable with a small number of recipients, it would not scale well to the case of hundreds or thousands of receivers; the amount of addressing information in the datagram would swamp the amount of data actually carried in the packet's payload field. Explicit identification of the receivers by the sender also requires that the sender know the identities and addresses of all of the receivers. We will see shortly that there are cases where this requirement might be undesirable.

For these reasons, in the Internet architecture (and other network architectures such as ATM [Black 1995]), a multicast packet is addressed using **address indirection**. That is, a single identifier is used for the group of receivers, and a copy of the packet that is addressed to the group using this single identifier is delivered to all of the multicast receivers associated with that group. In the Internet, the single identifier that represents a group of receivers is a class D multicast IP address. The group of receivers associated with a class D address is referred to as a **multicast group**. The multicast group abstraction is illustrated in Figure 4.48. Here, four hosts (shown in shaded color) are associated with the multicast group address of 226.17.30.197 and will receive all datagrams addressed to that multicast address. The difficulty that we must still address is the fact that each host has a unique IP unicast address that is completely independent of the address of the multicast group in which it is participating.

While the multicast group abstraction is simple, it raises a host (pun intended) of questions. How does a group get started and how does it terminate? How is the group address chosen? How are new hosts added to the group (either as senders or receivers)? Can anyone join a group (and send to, or receive from, that group) or is group membership restricted and, if so, by whom? Do group members know the identities of the other group members as part of the network-layer protocol? How do the network nodes interoperate with each other to deliver a multicast datagram to

DELL - Exhibit 1008 - Page 192

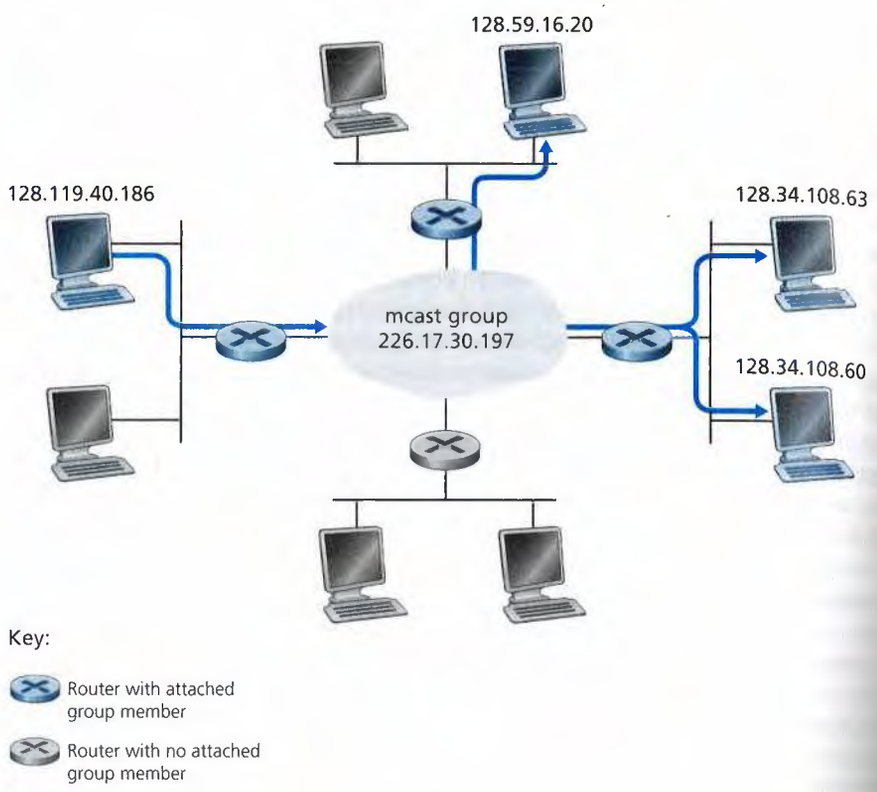


Figure 4.48 ♦ The multicast group: A datagram addressed to the group is delivered to all members of the multicast group.

all group members? For the Internet, the answers to all of these questions involve the Internet Group Management Protocol [RFC 3376]. So, let us next briefly consider IGMP and then return to these broader questions.

Internet Group Management Protocol

The IGMP protocol version 3 [RFC 3376] operates between a host and its directly attached router (informally, we can think of the directly attached router as the first-hop router that a host would see on a path to any other host outside its own local network, or the last-hop router on any path to that host), as shown in Figure 4.49. Figure 4.49 shows three first-hop multicast routers, each connected to its attached hosts via one outgoing local interface. This local interface is attached to a LAN in this example, and while each LAN has multiple attached hosts, at most a few of these hosts will typically belong to a given multicast group at any given time.

DELL - Exhibit 1008 - Page 193

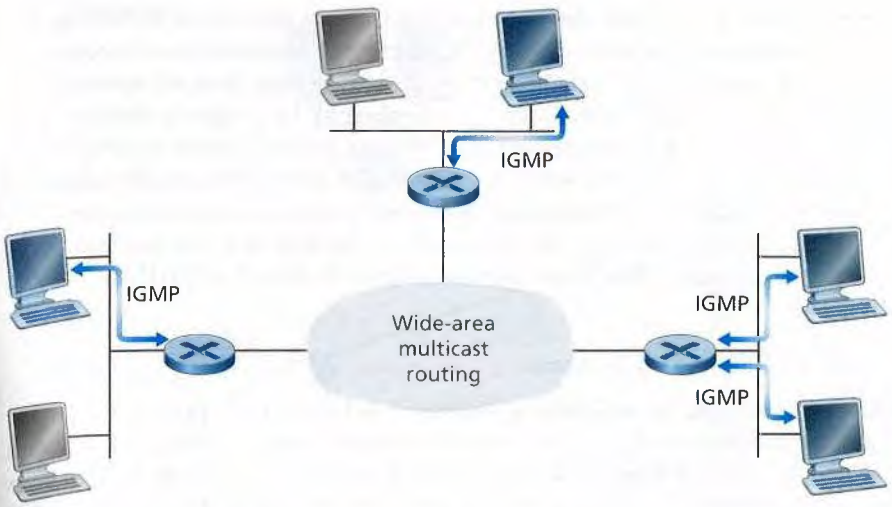


Figure 4.49 ♦ The two components of network-layer multicast in the Internet: IGMP and multicast routing protocols

IGMP provides the means for a host to inform its attached router that an application running on the host wants to join a specific multicast group. Given that the scope of IGMP interaction is limited to a host and its attached router, another protocol is clearly required to coordinate the multicast routers (including the attached routers) throughout the Internet, so that multicast datagrams are routed to their final destinations. This latter functionality is accomplished by network-layer multicast routing algorithms, such as those we will consider shortly. Network-layer multicast in the Internet thus consists of two complementary components: IGMP and multicast routing protocols.

IGMP has only three message types. Like ICMP, IGMP messages are carried (encapsulated) within an IP datagram, with an IP protocol number of 2. The `membership_query` message is sent by a router to all hosts on an attached interface (for example, to all hosts on a local area network) to determine the set of all multicast groups that have been joined by the hosts on that interface. Hosts respond to a `membership_query` message with an IGMP `membership_report` message. `membership_report` messages can also be generated by a host when an application first joins a multicast group without waiting for a `membership_query` message from the router. The final type of IGMP message is the `leave_group` message. Interestingly, this message is optional. But if it is optional, how does a router detect when a host leaves the multicast group? The answer to this question is that the router *infers* that a host is no longer in the multicast group if it no longer responds to a `membership_query` message with the given group address. This is an example of what is sometimes called **soft state** in an

DELL - Exhibit 1008 - Page 194

Internet protocol. In a soft-state protocol, the state (in this case of IGMP, the fact that there are hosts joined to a given multicast group) is removed via a timeout event (in this case, via a periodic `membership_query` message from the router) if it is not explicitly refreshed (in this case, by a `membership_report` message from an attached host). It has been argued that soft-state protocols result in simpler control than hard-state protocols, which not only require state to be explicitly added and removed, but also require mechanisms to recover from the situation where the entity responsible for removing state has terminated prematurely or failed. Interesting discussions of soft state can be found in [Raman 1999; Ji 2003; Lui 2004].

Multicast Routing Algorithms

The **multicast routing problem** is illustrated in Figure 4.50. Hosts joined to the multicast group are shaded in color; their immediately attached router is also shaded in color. As shown in Figure 4.50, only a subset of routers (those with attached hosts that are joined to the multicast group) actually needs to receive the multicast traffic. In Figure 4.50, only routers *A*, *B*, *E*, and *F* need to receive the multicast traffic. Since none of the hosts attached to router *D* are joined to the multicast group and since router *C* has no attached hosts, neither *C* nor *D* needs to receive the multicast group traffic. The goal of multicast routing, then, is to find a tree of links that connects all of the routers that have attached hosts belonging to the multicast group. Multicast packets will then be routed along this tree from the sender to all of the hosts belonging to the multicast tree. Of course, the tree may contain routers that do not have attached hosts belonging to the multicast group (for example, in Figure 4.50, it is impossible to connect routers *A*, *B*, *E*, and *F* in a tree without involving either router *C* or *D*).

In practice, two approaches have been adopted for determining the multicast routing tree, both of which we have already studied in the context of broadcast routing, and so we will only mention them in passing here. The two approaches differ according to whether a single group-shared tree is used to distribute the traffic for *all* senders in the group, or whether a source-specific routing tree is constructed for each individual sender.

- *Multicast routing using a group-shared tree.* As in the case of spanning-tree broadcast, multicast routing over a group-shared tree is based on building a tree that includes all edge routers with attached hosts belonging to the multicast group. In practice, a center-based approach is used to construct the multicast routing tree, with edge routers with attached hosts belonging to the multicast group sending (via unicast) join messages addressed to the center node. As in the broadcast case, a join message is forwarded using unicast routing toward the center until it either arrives at a router that already belongs to the multicast tree or arrives at the center. All routers along the path that the join message follows will then forward received multicast packets to the edge router that

DELL - Exhibit 1008 - Page 195

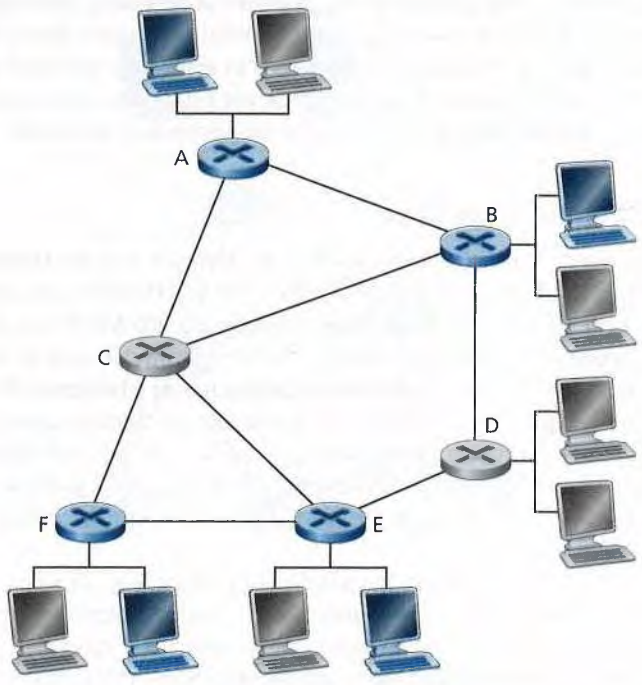


Figure 4.50 ♦ Multicast hosts, their attached routers, and other routers

initiated the multicast join. A critical question for center-based tree multicast routing is the process used to select the center. Center-selection algorithms are discussed in [Wall 1980; Thaler 1997; Estrin 1997].

- *Multicast routing using a source-based tree.* While group-shared tree multicast routing constructs a single, shared routing tree to route packets from *all* senders, the second approach constructs a multicast routing tree for *each* source in the multicast group. In practice, an RPF algorithm (with source node x) is used to construct a multicast forwarding tree for multicast datagrams originating at source x. The RPF broadcast algorithm we studied earlier requires a bit of tweaking for use in multicast. To see why, consider router D in Figure 4.51. Under broadcast RPF, it would forward packets to router G, even though router G has no attached hosts that are joined to the multicast group. While this is not so bad for this case where D has only a single downstream router, G, imagine what would happen if there were thousands of routers downstream from D! Each of these thousands of routers would receive unwanted multicast packets. (This scenario is not as far-fetched as it might seem. The initial MBone [Casner 1992; Macedonia 1994], the first global multicast network, suffered from precisely this

DELL - Exhibit 1008 - Page 196

problem at first.). The solution to the problem of receiving unwanted multicast packets under RPF is known as **pruning**. A multicast router that receives multicast packets and has no attached hosts joined to that group will send a prune message to its upstream router. If a router receives prune messages from each of its downstream routers, then it can forward a prune message upstream.

Multicast Routing in the Internet

The first multicast routing protocol used in the Internet was the **Distance-Vector Multicast Routing Protocol (DVMRP)** [RFC 1075]. DVMRP implements source-based trees with reverse path forwarding and pruning. DVMRP uses an RPF algorithm with pruning, as discussed above. Perhaps the most widely used Internet multicast routing protocol is the **Protocol-Independent Multicast (PIM) routing protocol**, which explicitly recognizes two multicast distribution scenarios. In dense mode [RFC 3973], multicast group members are densely located; that is, many or most of the routers in the area need to be involved in routing multicast datagrams. PIM dense mode is a flood-and-prune reverse path forwarding technique similar in spirit to DVMRP.

In sparse mode [RFC 4601], the number of routers with attached group members is small with respect to the total number of routers; group members are widely dispersed. PIM sparse mode uses rendezvous points to set up the multicast distribution tree. In **source-specific multicast (SSM)** [RFC 3569, RFC 4607], only a single sender is allowed to send traffic into the multicast tree, considerably simplifying tree construction and maintenance.

When PIM and DVMP are used within a domain, the network operator can configure IP multicast routers within the domain, in much the same way that intra-domain unicast routing protocols such as RIP, IS-IS, and OSPF can be configured. But what happens when multicast routes are needed between different domains? Is there a multicast equivalent of the inter-domain BGP protocol? The answer is (literally) yes. [RFC 4271] defines multiprotocol extensions to BGP to allow it to carry routing information for other protocols, including multicast information. The Multicast Source Discovery Protocol (MSDP) [RFC 3618, RFC 4611] can be used to connect together rendezvous points in different PIM sparse mode domains.

Let us close our discussion of IP multicast by noting that IP multicast has yet to take off in a big way. For interesting discussions of the current Internet multicast service model and deployment issues, see [Diot 2000, Sharma 2003]. Nonetheless, in spite of the lack of widespread deployment, network-level multicast is far from "dead." Multicast traffic has been carried for many years on Internet 2, and the networks with which it peers [Internet2 Multicast 2007]. In the United Kingdom, the BBC is engaged in trials of content distribution via IP multicast [BBC Multicast 2007]. At the same time, application-level multicast, as we saw with PPLive in Chapter 2 and in other peer-to-peer systems such as End System Multicast [ESM 2007], provides multicast distribution of content among peers using application-layer (rather

DELL - Exhibit 1008 - Page 197

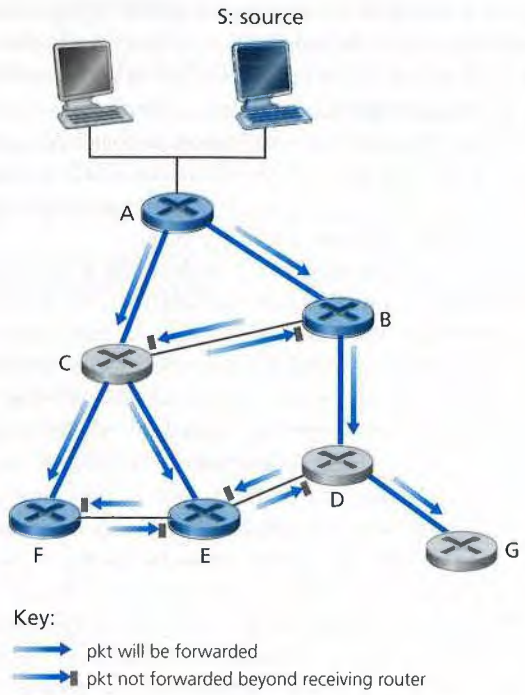


Figure 4.51 ♦ Reverse path forwarding, the multicast case

than network-layer) multicast protocols. Will future multicast services be primarily implemented in the network layer (in the network core) or in the application layer (at the network's edge)? While the current craze for content distribution via peer-to-peer approaches tips the balance in favor of application-layer multicast at least in the near-term future, progress continues to be made in IP multicast, and sometimes the race ultimately goes to the slow and steady.

4.8 Summary

In this chapter, we began our journey into the network core. We learned that the network layer involves each and every host and router in the network. Because of this, network-layer protocols are among the most challenging in the protocol stack.

We learned that a router may need to process millions of flows of packets between different source-destination pairs at the same time. To permit a router to process such a large number of flows, network designers have learned over the years

DELL - Exhibit 1008 - Page 198

that the router's tasks should be as simple as possible. Many measures can be taken to make the router's job easier, including using a datagram network layer rather than a virtual-circuit network layer, using a streamlined and fixed-sized header (as in IPv6), eliminating fragmentation (also done in IPv6), and providing the one and only best-effort service. Perhaps the most important trick here is *not* to keep track of individual flows, but instead base routing decisions solely on hierarchically structured destination addresses in the datagrams. It is interesting to note that the postal service has been using this approach for many years.

In this chapter, we also looked at the underlying principles of routing algorithms. We learned how routing algorithms abstract the computer network to a graph with nodes and links. With this abstraction, we can exploit the rich theory of shortest-path routing in graphs, which has been developed over the past 40 years in the operations research and algorithms communities. We saw that there are two broad approaches: a centralized (global) approach, in which each node obtains a complete map of the network and independently applies a shortest-path routing algorithm; and a decentralized approach, in which individual nodes have only a partial picture of the entire network, yet the nodes work together to deliver packets along the shortest routes. We also studied how hierarchy is used to deal with the problem of scale by partitioning large networks into independent administrative domains called autonomous systems (ASs). Each AS independently routes its datagrams through the AS, just as each country independently routes its postal mail through the country. We learned how centralized, decentralized, and hierarchical approaches are embodied in the principal routing protocols in the Internet: RIP, OSPF, and BGP. We concluded our study of routing algorithms by considering broadcast and multicast routing.

Having completed our study of the network layer, our journey now takes us one step further down the protocol stack, namely, to the link layer. Like the network layer, the link layer is also part of the network core. But we will see in the next chapter that the link layer has the much more localized task of moving packets between nodes on the same link or LAN. Although this task may appear on the surface to be trivial compared with that of the network layer's tasks, we will see that the link layer involves a number of important and fascinating issues that can keep us busy for a long time.



Homework Problems and Questions

Chapter 4 Review Questions

SECTIONS 4.1–4.2

R1. Let's review some of the terminology used in this textbook. Recall that the name of a transport-layer packet is *segment* and that the name of a link-layer packet is *frame*. What is the name of a network-layer packet? Recall that both

DELL - Exhibit 1008 - Page 199

routers and link-layer switches are called *packet switches*. What is the fundamental difference between a router and link-layer switch? Recall that we use the term *routers* for both datagram networks and VC networks.

R2. What are the two most important network-layer functions in a datagram network? What are the three most important network-layer functions in a virtual-circuit network?

R3. What is the difference between routing and forwarding?

R4. Do the routers in both datagram networks and virtual-circuit networks use forwarding tables? If so, describe the forwarding tables for both classes of networks.

R5. Describe some hypothetical services that the network layer can provide to a single packet. Do the same for a flow of packets. Are any of your hypothetical services provided by the Internet's network layer? Are any provided by ATM's CBR service model? Are any provided by ATM's ABR service model?

R6. List some applications that would benefit from ATM's CBR service model.

SECTION 4.3

R7. Discuss why each input port in a high-speed router stores a shadow copy of the forwarding table.

R8. Three types of switching fabrics are discussed in Section 4.3. List and briefly describe each type.

R9. Describe how packet loss can occur at input ports. Describe how packet loss at input ports can be eliminated (without using infinite buffers).

R10. Describe how packet loss can occur at output ports.

R11. What is HOL blocking? Does it occur in input ports or output ports?

SECTION 4.4

R12. Do routers have IP addresses? If so, how many?

R13. What is the 32-bit binary equivalent of the IP address 223.1.3.27?

R14. Visit a host that uses DHCP to obtain its IP address, network mask, default router, and IP address of its local DNS server. List these values.

R15. Suppose there are three routers between a source host and a destination host. Ignoring fragmentation, an IP datagram sent from the source host to the destination host will travel over how many interfaces? How many forwarding tables will be indexed to move the datagram from the source to the destination?

R16. Suppose an application generates chunks of 40 bytes of data every 20 msec, and each chunk gets encapsulated in a TCP segment and then an IP datagram.

DELL - Exhibit 1008 - Page 200

What percentage of each datagram will be overhead, and what percentage will be application data?

R17. Suppose Host A sends Host B a TCP segment encapsulated in an IP datagram. When Host B receives the datagram, how does the network layer in Host B know it should pass the segment (that is, the payload of the datagram) to TCP rather than to UDP or to something else?

R18. Suppose you purchase a wireless router and connect it to your cable modem. Also suppose that your ISP dynamically assigns your connected device (that is, your wireless router) one IP address. Also suppose that you have five PCs at home that use 802.11 to wirelessly connect to your wireless router. How are IP addresses assigned to the five PCs? Does the wireless router use NAT? Why or why not?

R19. Compare and contrast the IPv4 and the IPv6 header fields. Do they have any fields in common?

R20. It has been said that when IPv6 tunnels through IPv4 routers, IPv6 treats the IPv4 tunnels as link-layer protocols. Do you agree with this statement? Why or why not?

SECTION 4.5

R21. Compare and contrast link-state and distance-vector routing algorithms.

R22. Discuss how a hierarchical organization of the Internet has made it possible to scale to millions of users.

R23. Is it necessary that every autonomous system use the same intra-AS routing algorithm? Why or why not?

SECTION 4.6

R24. Consider Figure 4.35. Starting with the original table in *D*, suppose that *D* receives from *A* the following advertisement:

Destination Subnet	Next Router	Number of Hops to Destination
z	C	10
w	—	1
x	—	1
....	....	....

Will the table in *D* change? If so how?

R25. Compare and contrast the advertisements used by RIP and OSPF.

R26. Fill in the blank: RIP advertisements typically announce the number of hops to various destinations. BGP updates, on the other hand, announce the _____ to the various destinations.

DELL - Exhibit 1008 - Page 201

R27. Why are different inter-AS and intra-AS protocols used in the Internet?

R28. Why are policy considerations as important for intra-AS protocols, such as OSPF and RIP, as they are for an inter-AS routing protocol like BGP?

R29. Define and contrast the following terms: *subnet*, *prefix*, and *BGP route*.

R30. How does BGP use the NEXT-HOP attribute? How does it use the AS-PATH attribute?

R31. Describe how a network administrator of an upper-tier ISP can implement policy when configuring BGP.

SECTION 4.7

R32. What is an important difference between implementing the broadacst abstraction via multiple unicasts, and a single network- (router-) supported broadcast?

R33. For each of the three general approaches we studied for broadcast communication (uncontrolled flooding, controlled flooding, and spanning-tree broadcast), are the following statements true or false? You may assume that no packets are lost due to buffer overflow and all packets are delivered on a link in the order in which they were sent.

a. A node may receive multiple copies of the same packet.

b. A node may forward multiple copies of a packet over the same outgoing link.

R34. When a host joins a multicast group, must it change its IP address to that of the multicast group it is joining?

R35. What are the roles played by the IGMP protocol and a wide-area multicast routing protocol?

R36. What is the difference between a group-shared tree and a source-based tree in the context of multicast routing?

Problems

P1. Consider some of the pros and cons of virtual-circuit and datagram networks.

a. Suppose that in the network layer, routers were subjected to stressful conditions that might cause them to fail fairly often. At a high level, what actions would need to be taken on such router failure? Does this argue in favor of VC or datagram architecture?

b. Suppose that in order to provide a guarantee regarding the level of performance (for example, delay) that would be seen along a source-to-destination

DELL - Exhibit 1008 - Page 202

path, the network requires a sender to declare its peak traffic rate. If the declared peak traffic rate and the existing declared traffic rates are such that there is no way to get traffic from the source to the destination that meets the required delay requirements, the source is not allowed access to the network. Would such an approach be more easily accomplished within a VC or a datagram architecture?

P2. Consider a virtual-circuit network. Suppose the VC number is a 16-bit field.

a. What is the maximum number of virtual circuits that can be carried over a link?

b. Suppose a central node determines paths and VC numbers at connection setup. Suppose the same VC number is used on each link along the VC's path. Describe how the central node might determine the VC number at connection setup. Is it possible that there are fewer VCs in progress than the maximum as determined in part (a) yet there is no common free VC number?

c. Suppose that different VC numbers are permitted in each link along a VC's path. During connection setup, after an end-to-end path is determined, describe how the links can choose their VC numbers and configure their forwarding tables in a decentralized manner, without reliance on a central node.

P3. A bare-bones forwarding table in a VC network has four columns. What is the meaning of the values in each of these columns? A bare-bones forwarding table in a datagram network has two columns. What is the meaning of the values in each of these columns?

P4. Consider a VC network with a 2-bit field for the VC number. Suppose that the network wants to set up a virtual circuit over four links: link A, link B, link C, and link D. Suppose that each of these links is currently carrying two other virtual circuits, and the VC numbers of these other VCs are as follows:

Link A	Link B	Link C	Link D
00	01	10	11
01	10	11	00

In answering the following questions, keep in mind that each of the existing VCs may only be traversing one of the four links.

a. If each VC is required to use the same VC number on all links along its path, what VC number could be assigned to the new VC?

DELL - Exhibit 1008 - Page 203

b. If each VC is permitted to have different VC numbers in the different links along its path (so that forwarding tables must perform VC number translation), how many different combinations of four VC numbers (one for each of the four links) could be used?

P5. In the text we have used the term *connection-oriented service* to describe a transport-layer service and *connection service* for a network-layer service. Why the subtle shades in terminology?

P6. In Section 4.3, we noted that there can be no input queuing if the switching fabric is n times faster than the input line rates, assuming n input lines all have the same line rate. Explain (in words) why this should be so.

P7. Consider a router with a switch fabric, 2 input ports (A and B) and 2 output ports (C and D). Suppose the switch fabric operates at 1.5 times the line speed.

a. If, for some reason, all packets from A are destined to D, and all packets from B are destined to C, can a switch fabric be designed so that there is no input port queuing? Explain why or why not in one sentence.

b. Suppose now packets from A and B are randomly destined to both C and D. Can a switch fabric be designed so that there is no input port queuing? Explain why or why not in one sentence.

P8. Consider a datagram network using 32-bit host addresses. Suppose a router has four links, numbered 0 through 3, and packets are to be forwarded to the link interfaces as follows:

Destination Address Range	Link Interface
11100000 00000000 00000000 00000000 through 11100000 11111111 11111111 11111111	0
11100001 00000000 00000000 00000000 through 11100001 00000000 11111111 11111111	1
11100001 00000001 00000000 00000000 through 11100001 11111111 11111111 11111111	2
otherwise	3

a. Provide a forwarding table that has four entries, uses longest prefix matching, and forwards packets to the correct link interfaces.

DELL - Exhibit 1008 - Page 204

b. Describe how your forwarding table determines the appropriate link interface for datagrams with destination addresses:

11001000 10010001 01010001 01010101
11100001 00000000 11000011 00111100
11100001 10000000 00010001 01110111

P9. Consider a datagram network using 8-bit host addresses. Suppose a router uses longest prefix matching and has the following forwarding table:

Prefix Match	Interface
00	0
01	1
10	2
11	3

For each of the four interfaces, give the associated range of destination host addresses and the number of addresses in the range.

P10. Consider a datagram network using 8-bit host addresses. Suppose a router uses longest prefix matching and has the following forwarding table:

Prefix Match	Interface
1	0
11	1
111	2
otherwise	3

For each of the four interfaces, give the associated range of destination host addresses and the number of addresses in the range.

P11. Consider a router that interconnects three subnets: Subnet 1, Subnet 2, and Subnet 3. Suppose all of the interfaces in each of these three subnets are required to have the prefix 223.1.17/24. Also suppose that Subnet 1 is required to support up to 125 interfaces, and Subnets 2 and 3 are each required to support up to 60 interfaces. Provide three network addresses (of the form a.b.c.d/x) that satisfy these constraints.

P12. In Section 4.2.2 an example forwarding table (using longest prefix matching) is given. Rewrite this forwarding table using the a.b.c.d/x notation instead of the binary string notation.

DELL - Exhibit 1008 - Page 205

P13. In Problem P7 you are asked to provide a forwarding table (using longest prefix matching). Rewrite this forwarding table using the a.b.c.d/x notation instead of the binary string notation.

P14. Consider a subnet with prefix 101.101.101.64/26. Give an example of one IP address (of form xxx.xxx.xxx.xxx) that can be assigned to this network. Suppose an ISP owns the block of addresses of the form 101.101.128/17. Suppose it wants to create four subnets from this block, with each block having the same number of IP addresses. What are the prefixes (of form a.b.c.d/x) for the four subnets?

P15. Consider the topology shown in Figure 4.17. Denote the three subnets with hosts (starting clockwise at 12:00) as Networks A, B, and C. Denote the subnets without hosts as Networks D, E, and F.

a. Assign network addresses to each of these six subnets, with the following constraints: All addresses must be allocated from 214.97.254/23; Subnet A should have enough addresses to support 250 interfaces; Subnet B should have enough addresses to support 120 interfaces; and Subnet C should have enough addresses to support 120 interfaces. Of course, subnets D, E and F should each be able to support two interfaces. For each subnet, the assignment should take the form a.b.c.d/x or a.b.c.d/x – e.f.g.h/y.

b. Using your answer to part (a), provide the forwarding tables (using longest prefix matching) for each of the three routers.

P16. Consider sending a 3,000-byte datagram into a link that has an MTU of 500 bytes. Suppose the original datagram is stamped with the identification number 422. How many fragments are generated? What are their characteristics?

P17. Suppose datagrams are limited to 1,500 bytes (including header) between source Host A and destination Host B. Assuming a 20-byte IP header, how many datagrams would be required to send an MP3 consisting of 4 million bytes?

P18. Consider the network setup in Figure 4.22. Suppose that the ISP instead assigns the router the address 126.13.89.67 and that the network address of the home network is 192.168/16.

a. Assign addresses to all interfaces in the home network.

b. Suppose each host has two ongoing TCP connections, all to port 80 at host 128.119.40.86. Provide the six corresponding entries in the NAT translation table.

P19. In this problem we'll explore the impact of NATs on P2P applications. Suppose a peer with username Arnold discovers through querying that a peer with username Bernard has a file it wants to download. Also suppose that

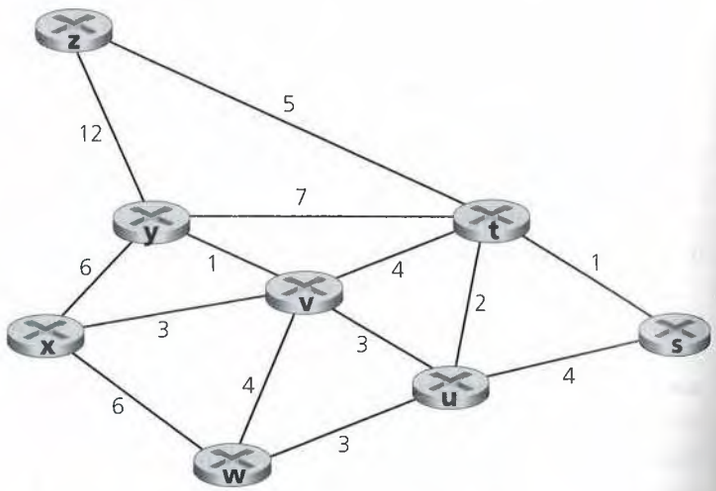
DELL - Exhibit 1008 - Page 206

Bernard and Arnold are both behind a NAT. Try to devise a technique that will allow Arnold to establish a TCP connection with Bernard without application-specific NAT configuration. If you have difficulty devising such a technique, discuss why.

P20. Looking at Figure 4.27, enumerate the paths from v to y that do not contain any loops.

P21. Repeat problem P20 for paths from x to w, w to u, and z to x.

P22. Consider the following network. With the indicated link costs, use Dijkstra's shortest-path algorithm to compute the shortest path from x to all network nodes. Show how the algorithm works by computing a table similar to Table 4.3.



P23. Consider the network shown in problem P22. Using Dijkstra's algorithm, and showing your work using a table similar to Table 4.3, do the following:

a. Compute the shortest path from s to all network nodes.

b. Compute the shortest path from t to all network nodes.

c. Compute the shortest path from u to all network nodes.

d. Compute the shortest path from v to all network nodes.

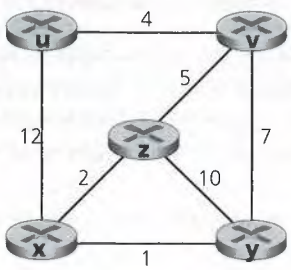
e. Compute the shortest path from w to all network nodes.

f. Compute the shortest path from y to all network nodes.

g. Compute the shortest path from z to all network nodes.

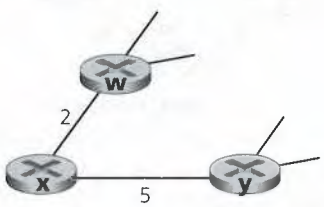
DELL - Exhibit 1008 - Page 207

P24. Consider the network shown below, and assume that each node initially knows the costs to each of its neighbors. Consider the distance-vector algorithm and show the distance table entries at node z.



P25. Consider a general topology (that is, not the specific network shown above) and a synchronous version of the distance-vector algorithm. Suppose that at each iteration, a node exchanges its distance vectors with its neighbors and receives their distance vectors. Assuming that the algorithm begins with each node knowing only the costs to its immediate neighbors, what is the maximum number of iterations required before the distributed algorithm converges? Justify your answer.

P26. Consider the network fragment shown below. x has only two attached neighbors, w and y. w has a minimum-cost path to destination u (not shown) of 5, and y has a minimum-cost path to u of 6. The complete paths from w and y to u (and between w and y) are not shown. All link costs in the network have strictly positive integer values.



a. Give x's distance vector for destinations w, y, and u.

b. Give a link-cost change for either $c(x,w)$ or $c(x,y)$ such that x will inform its neighbors of a new minimum-cost path to u as a result of executing the distance-vector algorithm.

c. Give a link-cost change for either $c(x,w)$ or $c(x,y)$ such that x will *not* inform its neighbors of a new minimum-cost path to u as a result of executing the distance-vector algorithm.

P27. Consider the three-node topology shown in Figure 4.30. Rather than having the link costs shown in Figure 4.30, the link costs are $c(x,y) = 5$, $c(y,z) = 6$,

DELL - Exhibit 1008 - Page 208

$c(z,x) = 2$. Compute the distance tables after the initialization step and after each iteration of a synchronous version of the distance-vector algorithm (as we did in our earlier discussion of Figure 4.30).

P28. Describe how loops in paths can be detected in BGP.

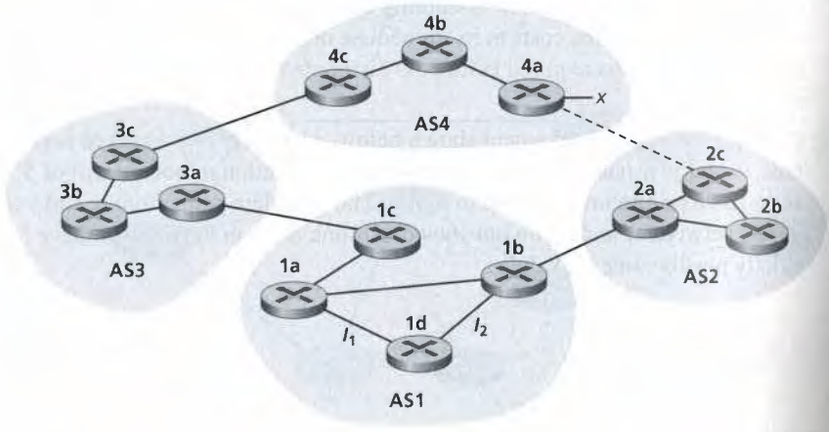
P29. Consider the network shown below. Suppose AS3 and AS2 are running OSPF for their intra-AS routing protocol. Suppose AS1 and AS4 are running RIP for their intra-AS routing protocol. Suppose eBGP and iBGP are used for the inter-AS routing protocol. Initially suppose there is *no* physical link between AS2 and AS4.

a. Router 3c learns about prefix x from which routing protocol: OSPF, RIP, eBGP or iBGP?

b. Router 3a learns about x from which routing protocol?

c. Router 1c learns about x from which routing protocol?

d. Router 1d learns about x from which routing protocol?



P30. Referring to the previous problem, once router 1d learns about x it will put an entry (x, l) in its forwarding table.

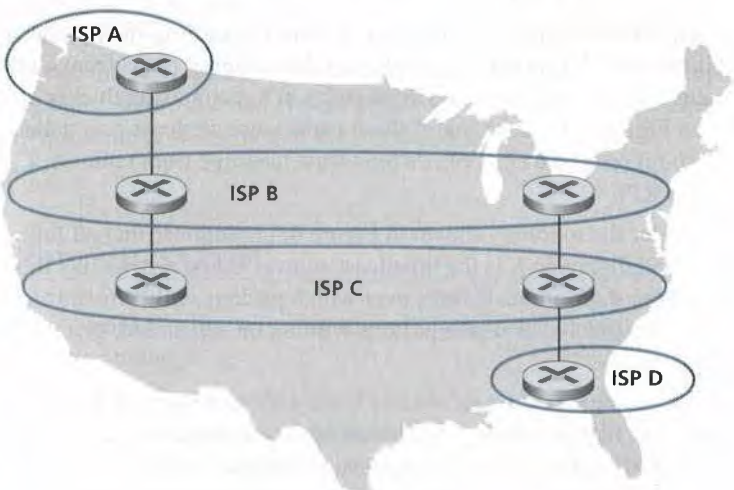
a. Will l be equal to l_1 or l_2 for this entry? Explain why in one sentence.

b. Now suppose that there is a physical link between AS2 and AS4, shown by the dotted line. Suppose router 1d learns that x is accessible via AS2 as well as via AS3. Will l be set to l_1 or l_2? Explain why in one sentence.

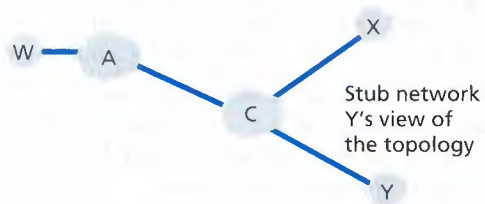
c. Now suppose there is another AS, called AS5, which lies on the path between AS2 and AS4 (not shown in diagram). Suppose router 1d learns that x is accessible via AS2 AS5 AS4 as well as via AS3 AS4. Will l be set to l_1 or l_2? Explain why in one sentence.

DELL - Exhibit 1008 - Page 209

P31. Consider the following network. ISP B provides national backbone service to regional ISP A. ISP C provides national backbone service to regional ISP D. Each ISP consists of one AS. B and C peer with each other in two places using BGP. Consider traffic going from A to D. B would prefer to hand that traffic over to C on the West Coast (so that C would have to absorb the cost of carrying the traffic cross-country), while C would prefer to get the traffic via its East Coast peering point with B (so that B would have carried the traffic across the country). What BGP mechanism might C use, so that B would hand over A-to-D traffic at its East Coast peering point? To answer this question, you will need to dig into the BGP specification.



P32. In Figure 4.43, consider the path information that reaches stub networks W, X, and Y. Based on the information available at W and X, what are their respective views of the network topology? Justify your answer. The topology view at Y is shown below.



P33. Consider the eight-node network (with nodes labeled s to z) in Problem P22. Show the minimal-cost tree rooted at s that includes (as end hosts) nodes u, v, w, and y. Informally argue why your tree is a minimal-cost tree.

DELL - Exhibit 1008 - Page 210

P34. Consider the two basic approaches identified for achieving broadcast, unicast emulation and network-layer (i.e., router-assisted) broadcast, and suppose spanning-tree broadcast is used to achive network-layer broadcast. Consider a single sender and 32 receivers. Suppose the sender is connected to the receivers by a binary tree of routers. What is the cost of sending a broadcast packet, in the cases of unicast emulation and network-layer broadcast, for this topology? Here, each time a packet (or copy of a packet) is sent over a single link, it incurs a unit of cost. What topology for interconnecting the sender, receivers, and routers will bring the cost of unicast emulation and true network-layer broadcast as far apart as possible? You can choose as many routers as you'd like.

P35. Consider the operation of the reverse path forwarding (RPF) algorithm in Figure 4.45. Using the same topology, find a set of paths from all nodes to the source node *A* (and indicate these paths in a graph using thicker-shaded lines as in Figure 4.45) such that if these paths were the least-cost paths, then node *B* would receive a copy of *A*'s broadcast message from nodes *A*, *C*, and *D* under RPF.

P36. Consider the topology shown in Figure 4.45. Suppose that all links have unit cost and that node *E* is the broadcast source. Using arrows like those shown in Figure 4.45) indicate links over which packets will be forwarded using RPF, and links over which packets will not be forwarded, given that node *E* is the source.

P37. Consider the topology shown in Figure 4.47, and suppose that each link has unit cost. Suppose node *C* is chosen as the center in a center-based multicast routing algorithm. Assuming that each attached router uses its least-cost path to node *C* to send join messages to *C*, draw the resulting center-based routing tree. Is the resulting tree a minimum-cost tree? Justify your answer.

P38. In Section 4.5.1 we studied Dijkstra's link-state routing algorithm for computing the unicast paths that are individually the least-cost paths from the source to all destinations. The union of these paths might be thought of as forming a **least-unicast-cost path tree** (or a shortest unicast path tree, if all link costs are identical). By constructing a counterexample, show that the least-cost path tree is *not* always the same as a minimum spanning tree.

P39. Consider a network in which all nodes are connected to three other nodes. In a single time step, a node can receive all transmitted broadcast packets from its neighbors, duplicate the packets, and send them to all of its neighbors (except to the node that sent a given packet). At the next time step, neighboring nodes can receive, duplicate, and forward these packets, and so on. Suppose that uncontrolled flooding is used to provide broadcast in such a network. At time step *t*, how many copies of the broadcast packet will be transmitted, assuming that during time step 1, a single broadcast packet is transmitted by the source node to its three neighbors.

DELL - Exhibit 1008 - Page 211

P40. We saw in Section 4.7 that there is no network-layer protocol that can be used to identify the hosts participating in a multicast group. Given this, how can multicast applications learn the identities of the hosts that are participating in a multicast group?

P41. Design (give a pseudocode description of) an application-level protocol that maintains the host addresses of all hosts participating in a multicast group. Specifically identify the network service (unicast or multicast) that is used by your protocol, and indicate whether your protocol is sending messages in-band or out-of-band (with respect to the application data flow among the multicast group participants) and why.

P42. What is the size of the multicast address space? Suppose now that two multicast groups randomly choose a multicast address. What is the probability that they choose the same address? Suppose now that 1,000 multicast groups are ongoing at the same time and choose their multicast group addresses at random. What is the probability that they interfere with each other?

Discussion Questions

D1. Find three companies that are currently selling high-speed router products. Compare the products.

D2. Use the whois service at the American Registry for Internet Numbers (http://www.arin.net/whois) to determine the IP address blocks for three universities. Can the whois services be used to determine with certainty the geographical location of a specific IP address?

D3. Is it possible to write the `ping` client program (using ICMP messages) in Java? Why or why not?

D4. In Section 4.4, we indicated that deployment of IPv6 has been slow. Why has it been slow? What is needed to accelerate its deployment?

D5. Discuss some of the problems NATs create for IPsec security (see [Phifer 2000]).

D6. Research the UPnP protocol. Specifically describe the messages that a host uses to reconfigure a NAT.

D7. Suppose ASs X and Z are not directly connected but instead are connected by AS Y. Further suppose that X has a peering agreement with Y, and that Y has a peering agreement with Z. Finally, suppose that Z wants to transit all of Y's traffic but does not want to transit X's traffic. Does BGP allow Z to implement this policy?

D8. In Section 4.7 we identified a number of multicast applications. Which of these applications are well suited for the minimalist Internet multicast service

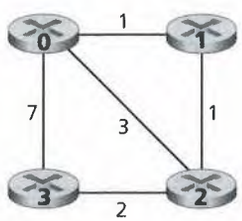
DELL - Exhibit 1008 - Page 212

model? Why? Which applications are not particularly well suited for this service model?

Programming Assignment

In this programming assignment, you will be writing a "distributed" set of procedures that implements a distributed asynchronous distance-vector routing for the network shown below.

You are to write the following routines that will "execute" asynchronously within the emulated environment provided for this assignment. For node 0, you will write the routines:



- *rtinit0()*. This routine will be called once at the beginning of the emulation. *rtinit0()* has no arguments. It should initialize your distance table in node 0 to reflect the direct costs of 1, 3, and 7 to nodes 1, 2, and 3, respectively. In the figure above, all links are bidirectional and the costs in both directions are identical. After initializing the distance table and any other data structures needed by your node 0 routines, it should then send its directly connected neighbors (in this case, 1, 2, and 3) the cost of its minimum-cost paths to all other network nodes. This minimum-cost information is sent to neighboring nodes in a routing update packet by calling the routine *tolayer2()*, as described in the full assignment. The format of the routing update packet is also described in the full assignment.
- *rtupdate0(struct rtpkt *rcvdpkt)*. This routine will be called when node 0 receives a routing packet that was sent to it by one of its directly connected neighbors. The parameter **rcvdpkt* is a pointer to the packet that was received. *rtupdate0()* is the "heart" of the distance-vector algorithm. The values it receives in a routing update packet from some other node *i* contain *i*'s current shortest-path costs to all other network nodes. *rtupdate0()* uses these received values to update its own distance table (as specified by the distance-vector algorithm). If its own minimum cost to another node changes as a result of the update, node 0 informs its directly connected neighbors of this change in minimum cost by sending them a routing packet. Recall that in the distance-vector

DELL - Exhibit 1008 - Page 213

algorithm, only directly connected nodes will exchange routing packets. Thus, nodes 1 and 2 will communicate with each other, but nodes 1 and 3 will not communicate with each other.

Similar routines are defined for nodes 1, 2, and 3. Thus, you will write eight procedures in all: *rtinit0()*, *rtinit1()*, *rtinit2()*, *rtinit3()*, *rtupdate0()*, *rtupdate1()*, *rtupdate2()*, and *rtupdate3()*. These routines will together implement a distributed, asynchronous computation of the distance tables for the topology and costs shown in the figure on the preceding page.

You can find the full details of the programming assignment, as well as C code that you will need to create the simulated hardware/software environment, at http://www.awl.com/kurose-ross. A Java version of the assignment is also available.

Ethereal Labs

In the companion Web site for this textbook, http://www.awl.com/kurose-ross, you'll find two Ethereal lab assignments. The first lab examines the operation of the IP protocol, and the IP datagram format in particular. The second lab explores the use of the ICMP protocol in the ping and traceroute commands.

DELL - Exhibit 1008 - Page 214

AN INTERVIEW WITH...

Vinton G. Cerf



Vinton G. Cerf is Vice President and Chief Internet Evangelist for Google. He served for over 16 years at MCI in various positions, ending up his tenure there as Senior Vice President for Technology Strategy. He is widely known as the co-designer of the TCP/IP protocols and the architecture of the Internet. During his time from 1976 to 1982 at the US Department of Defense Advanced Research Projects Agency (DARPA), he played a key role leading the development of Internet and Internet-related data packet and security techniques. He received the US Presidential Medal of Freedom in 2005 and the US National Medal of Technology in 1997. He holds a BS in Mathematics from Stanford University and an MS and PhD in computer science from UCLA.

What brought you to specialize in networking?

I was working as a programmer at UCLA in the late 1960s. My job was supported by the US Defense Advanced Research Projects Agency (called ARPA then, called DARPA now). I was working in the laboratory of Professor Leonard Kleinrock on the Network Measurement Center of the newly created ARPAnet. The first node of the ARPAnet was installed at UCLA on September 1, 1969. I was responsible for programming a computer that was used to capture performance information about the ARPAnet and to report this information back for comparison with mathematical models and predictions of the performance of the network.

Several of the other graduate students and I were made responsible for working on the so-called host-level protocols of the ARPAnet—the procedures and formats that would allow many different kinds of computers on the network to interact with each other. It was a fascinating exploration into a new world (for me) of distributed computing and communication.

Did you imagine that IP would become as pervasive as it is today when you first designed the protocol?

When Bob Kahn and I first worked on this in 1973, I think we were mostly very focused on the central question: how can we make heterogeneous packet networks interoperate with one another, assuming we cannot actually change the networks themselves. We hoped that we could find a way to permit an arbitrary collection of packet-switched networks to be interconnected in a transparent fashion, so that host computers could communicate end-to-end without having to do any translations in between. I think we knew that we were dealing with powerful and expandable technology but I doubt we had a clear image of what the world would be like with hundreds of millions of computers all interlinked on the Internet.

DELL - Exhibit 1008 - Page 215



What do you now envision for the future of networking and the Internet? What major challenges/obstacles do you think lie ahead in their development?

I believe the Internet itself and networks in general will continue to proliferate. Already there is convincing evidence that there will be billions of Internet-enabled devices on the Internet, including appliances like cell phones, refrigerators, personal digital assistants, home servers, televisions, as well as the usual array of laptops, servers, and so on. Big challenges include support for mobility, battery life, capacity of the access links to the network, and ability to scale the optical core of the network up in an unlimited fashion. Designing an interplanetary extension of the Internet is a project in which I am deeply engaged at the Jet Propulsion Laboratory. We will need to cut over from IPv4 [32-bit addresses] to IPv6 [128 bits]. The list is long!

Who has inspired you professionally?

My colleague Bob Kahn; my thesis advisor, Gerald Estrin; my best friend, Steve Crocker (we met in high school and he introduced me to computers in 1960!); and the thousands of engineers who continue to evolve the Internet today.

Do you have any advice for students entering the networking/Internet field?

Think outside the limitations of existing systems—imagine what might be possible; but then do the hard work of figuring out how to get there from the current state of affairs. Dare to dream: a half dozen colleagues and I at the Jet Propulsion Laboratory have been working on the design of an interplanetary extension of the terrestrial Internet. It may take decades to implement this, mission by mission, but to paraphrase: "A man's reach should exceed his grasp, or what are the heavens for?"

DELL - Exhibit 1008 - Page 216