

Computer Networking

A Top-Down Approach



sixth edition

KUROSE | ROSS

COMPUTER NETWORKING

SIXTH EDITION

A Top-Down Approach



JAMES F. KUROSE

University of Massachusetts, Amherst

KEITH W. ROSS

Polytechnic Institute of NYU

PEARSON

Boston Columbus Indianapolis New York San Francisco Upper Saddle River
Amsterdam Cape Town Dubai London Madrid Milan Munich Paris Montréal Toronto
Delhi Mexico City São Paulo Sydney Hong Kong Seoul Singapore Taipei Tokyo

Vice President and Editorial Director, ECS:

Marcia Horton

Editor in Chief: Michael Hirsch**Editorial Assistant:** Emma Snider**Vice President Marketing:** Patrice Jones**Marketing Manager:** Yez Alayan**Marketing Coordinator:** Kathryn Ferranti**Vice President and Director of Production:**

Vince O'Brien

Managing Editor: Jeff Holcomb**Senior Production Project Manager:**

Marilyn Lloyd

Manufacturing Manager: Nick Sklitsis**Operations Specialist:** Lisa McDowell**Art Director, Cover:** Anthony Gemmellaro**Art Coordinator:** Janet Theurer/

Theurer Briggs Design

Art Studio: Patrice Rossi Calkin/

Rossi Illustration and Design

Cover Designer: Liz Harasymcuk**Text Designer:** Joyce Cosentino Wells**Cover Image:** ©Fancy/Alamy**Media Editor:** Dan Sandin**Full-Service Vendor:** PreMediaGlobal**Senior Project Manager:** Andrea Stefanowicz**Printer/Binder:** Edwards Brothers**Cover Printer:** Lehigh-Phoenix Color

This book was composed in Quark. Basal font is Times. Display font is Berkeley.

Copyright © 2013, 2010, 2008, 2005, 2003 by Pearson Education, Inc., publishing as Addison-Wesley. All rights reserved. Manufactured in the United States of America. This publication is protected by Copyright, and permission should be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. To obtain permission(s) to use material from this work, please submit a written request to Pearson Education, Inc., Permissions Department, One Lake Street, Upper Saddle River, New Jersey 07458, or you may fax your request to 201-236-3290.

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

Library of Congress Cataloging-in-Publication Data

Kurose, James F.

Computer networking : a top-down approach / James F. Kurose, Keith W. Ross.—6th ed.

p. cm.

Includes bibliographical references and index.

ISBN-13: 978-0-13-285620-1

ISBN-10: 0-13-285620-4

1. Internet. 2. Computer networks. I. Ross, Keith W., 1956- II. Title.

TK5105.875.I57K88 2012

004.6—dc23

2011048215

10 9 8 7 6 5 4 3 2 1

PEARSON

ISBN-10: 0-13-285620-4

ISBN-13: 978-0-13-285620-1

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 Chair Professor and Head of the Computer Science Department at Polytechnic Institute of NYU. Before joining NYU-Poly in 2003, he was a professor at the University of Pennsylvania (13 years) and a professor at Eurecom Institute (5 years). He received a B.S.E.E from Tufts University, a M.S.E.E. from Columbia University, and a Ph.D. in Computer and Control Engineering from The University of Michigan. Keith Ross is also the founder and original CEO of Wimba, which develops online multimedia applications for elearning and was acquired by Blackboard in 2010.

Professor Ross's research interests are in security and privacy, social networks, peer-to-peer networking, Internet measurement, video streaming, content distribution networks, and stochastic modeling. He is an IEEE Fellow, recipient of the Infocom 2009 Best Paper Award, and recipient of 2011 and 2008 Best Paper Awards for Multimedia Communications (awarded by IEEE Communications Society). He has served on numerous journal editorial boards and conference program committees, including *IEEE/ACM Transactions on Networking*, *ACM SIGCOMM*, *ACM CoNext*, and *ACM Internet Measurement Conference*. He also has served as an advisor to the Federal Trade Commission on P2P file sharing.



This page intentionally left blank

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

A big THANKS to my professors, colleagues,
and students all over the world.
KWR

This page intentionally left blank

Preface

Welcome to the sixth edition of *Computer Networking: A Top-Down Approach*. Since the publication of the first edition 12 years ago, our book has been adopted for use at many hundreds of colleges and universities, translated into 14 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 Sixth 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 sixth 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, the sixth edition has been revised and updated substantially:

- The Companion Web site has been significantly expanded and enriched to include VideoNotes and interactive exercises, as discussed later in this Preface.
- In Chapter 1, the treatment of access networks has been modernized, and the description of the Internet ISP ecosystem has been substantially revised, accounting for the recent emergence of content provider networks, such as Google's. The presentation of packet switching and circuit switching has also been reorganized, providing a more topical rather than historical orientation.
- In Chapter 2, Python has replaced Java for the presentation of socket programming. While still explicitly exposing the key ideas behind the socket API, Python code is easier to understand for the novice programmer. Moreover, unlike Java, Python provides access to raw sockets, enabling students to build a larger variety of network applications. Java-based socket programming labs have been replaced with corresponding Python labs, and a new Python-based ICMP Ping lab has been added. As always, when material is retired from the book, such as Java-based socket programming material, it remains available on the book's Companion Web site (see following text).
- In Chapter 3, the presentation of one of the reliable data transfer protocols has been simplified and a new sidebar on TCP splitting, commonly used to optimize the performance of cloud services, has been added.
- In Chapter 4, the section on router architectures has been significantly updated, reflecting recent developments and practices in the field. Several new integrative sidebars involving DNS, BGP, and OSPF are included.

- Chapter 5 has been reorganized and streamlined, accounting for the ubiquity of switched Ethernet in local area networks and the consequent increased use of Ethernet in point-to-point scenarios. Also, a new section on data center networking has been added.
- Chapter 6 has been updated to reflect recent advances in wireless networks, particularly cellular data networks and 4G services and architecture.
- Chapter 7, which focuses on multimedia networking, has gone through a major revision. The chapter now includes an in-depth discussion of streaming video, including adaptive streaming, and an entirely new and modernized discussion of CDNs. A newly added section describes the Netflix, YouTube, and Kankan video streaming systems. The material that has been removed to make way for these new topics is still available on the Companion Web site.
- Chapter 8 now contains an expanded discussion on endpoint authentication.
- Significant new material involving end-of-chapter problems has been added. As with all previous editions, homework problems have been revised, added, and removed.

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++, Java, or Python (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

indeed works well. Nevertheless, we have found that 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 12 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 feedback we received from teachers and students alike have confirmed that this top-down approach has many advantages and does indeed work well pedagogically. 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 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 socket programming, service models, and protocols—important concepts that resurface in all subsequent layers. By providing socket programming examples in Python, we highlight the central ideas without confusing students with complex code. Undergraduates in electrical engineering and computer science should not have difficulty following the Python code.

An Internet Focus

Although we dropped the phrase “Featuring the Internet” from the title of this book with the fourth edition, this doesn’t mean that we dropped our focus on the Internet! Indeed, nothing could be further 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. 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 has been 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.

Teaching Networking Principles

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

Each new copy of this textbook includes six months of access to a Companion Web site for all book readers at <http://www.pearsonhighered.com/kurose-ross>, which includes:

- *Interactive learning material.* An important new component of the sixth edition is the significantly expanded online and interactive learning material. The book’s Companion Web site now contains VideoNotes—video presentations of

important topics throughout the book done by the authors, as well as walk-throughs of solutions to problems similar to those at the end of the chapter. We've also added Interactive Exercises that can create (and present solutions for) problems similar to selected end-of-chapter problems. Since students can generate (and view solutions for) an unlimited number of similar problem instances, they can work until the material is truly mastered. We've seeded the Web site with VideoNotes and online problems for chapters 1 through 5 and will continue to actively add and update this material over time. As in earlier editions, the Web site contains the interactive Java applets that animate many key networking concepts. The site also has 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 the new material in this edition, we've removed material on ATM networks and the RTSP protocol for multimedia. 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, which 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.
- *Wireshark labs.* One's understanding of network protocols can be greatly deepened by seeing them in action. The Web site provides numerous Wireshark assignments that enable students to actually observe the sequence of messages exchanged between two protocol entities. The Web site includes separate Wireshark labs on HTTP, DNS, TCP, UDP, IP, ICMP, Ethernet, ARP, WiFi, SSL, and on tracing all protocols involved in satisfying a request to fetch a web page. We'll continue to add new labs over time.

Pedagogical Features

We have each been teaching computer networking for more than 20 years. Together, we bring more than 50 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.

Supplements for Instructors

We provide a complete supplements package to aid instructors in teaching this course. This material can be accessed from Pearson’s Instructor Resource Center (<http://www.pearsonhighered.com/irc>). Visit the Instructor 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 have been completely updated with this sixth edition. 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 Wireshark labs. As noted earlier, we’ve introduced many new homework problems in the first five chapters of the book.

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. After completing Chapter 1, we recommend 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 first 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 students and instructors to e-mail us with 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 four editions. We've incorporated many of these suggestions into later editions of the book. We also encourage instructors to send us new homework problems (and solutions) that would complement the current homework problems. We'll post these on the instructor-only portion of the Web site. We also 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 us. If the applet (including notation and terminology) is appropriate, we'll be happy to include it on the text's Web site, with an appropriate reference to the applet's authors.

So, 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 fifth edition. We are also *very* thankful to the many 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)
Lichun Bao (University of California at Irvine)
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 (Technicolor Research)
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)
Edmundo A. de Souza e Silva (Federal University of Rio de Janeiro)
Philippe Decuetos (Eurécom Institute)
Christophe Diot (Technicolor Research)
Prithula Dhunghel (Akamai)
Deborah Estrin (University of California, Los Angeles)
Michalis Faloutsos (University of California at Riverside)
Wu-chi Feng (Oregon Graduate Institute)
Sally Floyd (ICIR, University of California at Berkeley)
Paul Francis (Max Planck Institute)
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 (NYU-Poly)
Yang Guo (Alcatel/Lucent Bell Labs)
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)
Cheng Huang (Microsoft Research)
Esther A. Hughes (Virginia Commonwealth University)
Van Jacobson (Xerox PARC)
Pinak Jain (former NYU-Poly student)
Jobin James (University of California at Riverside)
Sugih Jamin (University of Michigan)
Shivkumar Kalyanaraman (IBM Research, India)
Jussi Kangasharju (University of Helsinki)
Sneha Kasera (University of Utah)
Parviz Kermani (formerly of IBM Research)

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 (Bloomberg)
Miguel A. Labrador (University of South Florida)
Simon Lam (University of Texas)
Steve Lai (Ohio State University)
Tom LaPorta (Penn State University)
Tim-Berners Lee (World Wide Web Consortium)
Arnaud Legout (INRIA)
Lee Leitner (Drexel University)
Brian Levine (University of Massachusetts)
Chunchun Li (former NYU-Poly student)
Yong Liu (NYU-Poly)
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)
Jenni Moyer (Comcast)
Erich Nahum (IBM Research)
Christos Papadopoulos (Colorado State University)
Craig Partridge (BBN Technologies)
Radia Perlman (Intel)
Jitendra Padhye (Microsoft Research)
Vern Paxson (University of California at Berkeley)
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)
Pablo Rodriguez (Telefonica)
Sumit Roy (University of Washington)
Avi Rubin (Johns Hopkins University)
Dan Rubenstein (Columbia University)
Douglas Salane (John Jay College)
Despina Saporilla (Cisco Systems)
John Schanz (Comcast)

Henning Schulzrinne (Columbia University)
Mischa Schwartz (Columbia University)
Ardash Sethi (University of Delaware)
Harish Sethu (Drexel University)
K. Sam Shanmugan (University of Kansas)
Prashant Shenoy (University of Massachusetts)
Clay Shields (Georgetown University)
Subin Shrestra (University of Pennsylvania)
Bojie Shu (former NYU-Poly student)
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)
Di Wu (Sun Yat-sen University)
Shirley Wynn (NYU-Poly)
Raj Yavatkar (Intel)
Yechiam Yemini (Columbia University)
Ming Yu (State University of New York at Binghamton)
Ellen Zegura (Georgia Institute of Technology)
Honggang Zhang (Suffolk University)
Hui Zhang (Carnegie Mellon University)
Lixia Zhang (University of California at Los Angeles)
Meng Zhang (former NYU-Poly student)
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)

We also want to thank the entire Addison-Wesley team—in particular, Michael Hirsch, Marilyn Lloyd, and Emma Snider—who have done an absolutely outstanding job on this sixth edition (and who have put up with two very finicky authors who seem congenitally unable to meet deadlines!). Thanks also to our artists, Janet Theurer and Patrice Rossi Calkin, for their work on the beautiful figures in this book, and to Andrea Stefanowicz and her team at PreMediaGlobal 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.

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 Access Networks	12
1.2.2 Physical Media	18
1.3 The Network Core	22
1.3.1 Packet Switching	22
1.3.2 Circuit Switching	27
1.3.3 A Network of Networks	32
1.4 Delay, Loss, and Throughput in Packet-Switched Networks	35
1.4.1 Overview of Delay in Packet-Switched Networks	35
1.4.2 Queuing Delay and Packet Loss	39
1.4.3 End-to-End Delay	42
1.4.4 Throughput in Computer Networks	44
1.5 Protocol Layers and Their Service Models	47
1.5.1 Layered Architecture	47
1.5.2 Encapsulation	53
1.6 Networks Under Attack	55
1.7 History of Computer Networking and the Internet	60
1.7.1 The Development of Packet Switching: 1961–1972	60
1.7.2 Proprietary Networks and Internetworking: 1972–1980	62
1.7.3 A Proliferation of Networks: 1980–1990	63
1.7.4 The Internet Explosion: The 1990s	64
1.7.5 The New Millennium	65
1.8 Summary	66
Homework Problems and Questions	68
Wireshark Lab	78
Interview: Leonard Kleinrock	80

Chapter 2 Application Layer	83
2.1 Principles of Network Applications	84
2.1.1 Network Application Architectures	86
2.1.2 Processes Communicating	88
2.1.3 Transport Services Available to Applications	91
2.1.4 Transport Services Provided by the Internet	93
2.1.5 Application-Layer Protocols	96
2.1.6 Network Applications Covered in This Book	97
2.2 The Web and HTTP	98
2.2.1 Overview of HTTP	98
2.2.2 Non-Persistent and Persistent Connections	100
2.2.3 HTTP Message Format	103
2.2.4 User-Server Interaction: Cookies	108
2.2.5 Web Caching	110
2.2.6 The Conditional GET	114
2.3 File Transfer: FTP	116
2.3.1 FTP Commands and Replies	118
2.4 Electronic Mail in the Internet	118
2.4.1 SMTP	121
2.4.2 Comparison with HTTP	124
2.4.3 Mail Message Format	125
2.4.4 Mail Access Protocols	125
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 Distributed Hash Tables (DHTs)	151
2.7 Socket Programming: Creating Network Applications	156
2.7.1 Socket Programming with UDP	157
2.7.2 Socket Programming with TCP	163
2.8 Summary	168
Homework Problems and Questions	169
Socket Programming Assignments	179
Wireshark Labs: HTTP, DNS	181
Interview: Marc Andreessen	182

Chapter 3 Transport Layer 185

3.1	Introduction and Transport-Layer Services	186
3.1.1	Relationship Between Transport and Network Layers	186
3.1.2	Overview of the Transport Layer in the Internet	189
3.2	Multiplexing and Demultiplexing	191
3.3	Connectionless Transport: UDP	198
3.3.1	UDP Segment Structure	202
3.3.2	UDP Checksum	202
3.4	Principles of Reliable Data Transfer	204
3.4.1	Building a Reliable Data Transfer Protocol	206
3.4.2	Pipelined Reliable Data Transfer Protocols	215
3.4.3	Go-Back-N (GBN)	218
3.4.4	Selective Repeat (SR)	223
3.5	Connection-Oriented Transport: TCP	230
3.5.1	The TCP Connection	231
3.5.2	TCP Segment Structure	233
3.5.3	Round-Trip Time Estimation and Timeout	238
3.5.4	Reliable Data Transfer	242
3.5.5	Flow Control	250
3.5.6	TCP Connection Management	252
3.6	Principles of Congestion Control	259
3.6.1	The Causes and the Costs of Congestion	259
3.6.2	Approaches to Congestion Control	265
3.6.3	Network-Assisted Congestion-Control Example: ATM ABR Congestion Control	266
3.7	TCP Congestion Control	269
3.7.1	Fairness	279
3.8	Summary	283
	Homework Problems and Questions	285
	Programming Assignments	300
	Wireshark Labs: TCP, UDP	301
	Interview: Van Jacobson	302

Chapter 4 The Network Layer 305

4.1	Introduction	306
4.1.1	Forwarding and Routing	308
4.1.2	Network Service Models	310
4.2	Virtual Circuit and Datagram Networks	313
4.2.1	Virtual-Circuit Networks	314
4.2.2	Datagram Networks	317
4.2.3	Origins of VC and Datagram Networks	319

4.3	What's Inside a Router?	320
4.3.1	Input Processing	322
4.3.2	Switching	324
4.3.3	Output Processing	326
4.3.4	Where Does Queuing Occur?	327
4.3.5	The Routing Control Plane	331
4.4	The Internet Protocol (IP): Forwarding and Addressing in the Internet	331
4.4.1	Datagram Format	332
4.4.2	IPv4 Addressing	338
4.4.3	Internet Control Message Protocol (ICMP)	353
4.4.4	IPv6	356
4.4.5	A Brief Foray into IP Security	362
4.5	Routing Algorithms	363
4.5.1	The Link-State (LS) Routing Algorithm	366
4.5.2	The Distance-Vector (DV) Routing Algorithm	371
4.5.3	Hierarchical Routing	379
4.6	Routing in the Internet	383
4.6.1	Intra-AS Routing in the Internet: RIP	384
4.6.2	Intra-AS Routing in the Internet: OSPF	388
4.6.3	Inter-AS Routing: BGP	390
4.7	Broadcast and Multicast Routing	399
4.7.1	Broadcast Routing Algorithms	400
4.7.2	Multicast	405
4.8	Summary	412
	Homework Problems and Questions	413
	Programming Assignments	429
	Wireshark Labs: IP, ICMP	430
	Interview: Vinton G. Cerf	431

Chapter 5	The Link Layer: Links, Access Networks, and LANs	433
5.1	Introduction to the Link Layer	434
5.1.1	The Services Provided by the Link Layer	436
5.1.2	Where Is the Link Layer Implemented?	437
5.2	Error-Detection and -Correction Techniques	438
5.2.1	Parity Checks	440
5.2.2	Checksumming Methods	442
5.2.3	Cyclic Redundancy Check (CRC)	443
5.3	Multiple Access Links and Protocols	445
5.3.1	Channel Partitioning Protocols	448
5.3.2	Random Access Protocols	449
5.3.3	Taking-Turns Protocols	459
5.3.4	DOCSIS: The Link-Layer Protocol for Cable Internet Access	460

5.4	Switched Local Area Networks	461
5.4.1	Link-Layer Addressing and ARP	462
5.4.2	Ethernet	469
5.4.3	Link-Layer Switches	476
5.4.4	Virtual Local Area Networks (VLANs)	482
5.5	Link Virtualization: A Network as a Link Layer	486
5.5.1	Multiprotocol Label Switching (MPLS)	487
5.6	Data Center Networking	490
5.7	Retrospective: A Day in the Life of a Web Page Request	495
5.7.1	Getting Started: DHCP, UDP, IP, and Ethernet	495
5.7.2	Still Getting Started: DNS and ARP	497
5.7.3	Still Getting Started: Intra-Domain Routing to the DNS Server	498
5.7.4	Web Client-Server Interaction: TCP and HTTP	499
5.8	Summary	500
	Homework Problems and Questions	502
	Wireshark Labs: Ethernet and ARP, DHCP	510
	Interview: Simon S. Lam	511

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
6.3.5	Advanced Features in 802.11	542
6.3.6	Personal Area Networks: Bluetooth and Zigbee	544
6.4	Cellular Internet Access	546
6.4.1	An Overview of Cellular Network Architecture	547
6.4.2	3G Cellular Data Networks: Extending the Internet to Cellular Subscribers	550
6.4.3	On to 4G: LTE	553
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	578
	Wireshark Lab: IEEE 802.11 (WiFi)	583
	Interview: Deborah Estrin	584

Chapter 7 Multimedia Networking **587**

7.1	Multimedia Networking Applications	588
7.1.1	Properties of Video	588
7.1.2	Properties of Audio	590
7.1.3	Types of Multimedia Network Applications	591
7.2	Streaming Stored Video	593
7.2.1	UDP Streaming	595
7.2.2	HTTP Streaming	596
7.2.3	Adaptive Streaming and DASH	600
7.2.4	Content Distribution Networks	602
7.2.5	Case Studies: Netflix, YouTube, and Kankan	608
7.3	Voice-over-IP	612
7.3.1	Limitations of the Best-Effort IP Service	612
7.3.2	Removing Jitter at the Receiver for Audio	614
7.3.3	Recovering from Packet Loss	617
7.3.4	Case Study: VoIP with Skype	620
7.4	Protocols for Real-Time Conversational Applications	623
7.4.1	RTP	624
7.4.2	SIP	627
7.5	Network Support for Multimedia	632
7.5.1	Dimensioning Best-Effort Networks	634
7.5.2	Providing Multiple Classes of Service	636
7.5.3	Diffserv	648
7.5.4	Per-Connection Quality-of-Service (QoS) Guarantees: Resource Reservation and Call Admission	652
7.6	Summary	655
	Homework Problems and Questions	656
	Programming Assignment	666
	Interview: Henning Schulzrinne	668

Chapter 8 Security in Computer Networks **671**

8.1	What Is Network Security?	672
8.2	Principles of Cryptography	675
8.2.1	Symmetric Key Cryptography	676
8.2.2	Public Key Encryption	683

8.3	Message Integrity and Digital Signatures	688
8.3.1	Cryptographic Hash Functions	689
8.3.2	Message Authentication Code	691
8.3.3	Digital Signatures	693
8.4	End-Point Authentication	700
8.4.1	Authentication Protocol <i>ap1.0</i>	700
8.4.2	Authentication Protocol <i>ap2.0</i>	701
8.4.3	Authentication Protocol <i>ap3.0</i>	702
8.4.4	Authentication Protocol <i>ap3.1</i>	703
8.4.5	Authentication Protocol <i>ap4.0</i>	703
8.5	Securing E-Mail	705
8.5.1	Secure E-Mail	706
8.5.2	PGP	710
8.6	Securing TCP Connections: SSL	711
8.6.1	The Big Picture	713
8.6.2	A More Complete Picture	716
8.7	Network-Layer Security: IPsec and Virtual Private Networks	718
8.7.1	IPsec and Virtual Private Networks (VPNs)	718
8.7.2	The AH and ESP Protocols	720
8.7.3	Security Associations	720
8.7.4	The IPsec Datagram	721
8.7.5	IKE: Key Management in IPsec	725
8.8	Securing Wireless LANs	726
8.8.1	Wired Equivalent Privacy (WEP)	726
8.8.2	IEEE 802.11i	728
8.9	Operational Security: Firewalls and Intrusion Detection Systems	731
8.9.1	Firewalls	731
8.9.2	Intrusion Detection Systems	739
8.10	Summary	742
	Homework Problems and Questions	744
	Wireshark Lab: SSL	752
	IPsec Lab	752
	Interview: Steven M. Bellovin	753

Chapter 9 Network Management 755

9.1	What Is Network Management?	756
9.2	The Infrastructure for Network Management	760
9.3	The Internet-Standard Management Framework	764
9.3.1	Structure of Management Information: SMI	766
9.3.2	Management Information Base: MIB	770

9.3.3	SNMP Protocol Operations and Transport Mappings	772
9.3.4	Security and Administration	775
9.4	ASN.1	778
9.5	Conclusion	783
	Homework Problems and Questions	783
	Interview: Jennifer Rexford	786
	References	789
	Index	823

COMPUTER NETWORKING

SIXTH EDITION

A Top-Down Approach



This page intentionally left blank



Computer Networks and the Internet

Today's Internet is arguably the largest engineered system ever created by mankind, with hundreds of millions of connected computers, communication links, and switches; with billions of users who connect via laptops, tablets, and smartphones; and with an array of new Internet-connected devices such as sensors, Web cams, game consoles, picture frames, and even washing machines. Given that the Internet is so large and has so many diverse components and uses, is there any hope of understanding how it works? Are there guiding principles and structure that can provide a foundation for understanding such an amazingly large and complex system? And if so, is it possible that it actually could be both interesting *and* fun to learn about computer networks? Fortunately, the answers to all of these questions is a resounding YES! Indeed, it's our aim in this book to 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 a broad overview of computer networking and the Internet. Our goal here is to paint a broad picture and set the context for the rest of this book, to see the forest through the trees. We'll cover a lot of ground in this introductory chapter and discuss a lot of the pieces of a computer network, without losing sight of the big picture.

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 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 of data 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'll use the public Internet, a specific computer network, as our principal vehicle for discussing computer networks and their protocols. But what *is* the Internet? There are a couple of ways to answer this question. First, we can describe the nuts and bolts of the Internet, that is, the basic hardware and software components that make up the Internet. Second, we can 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 hundreds of 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 laptops, smartphones, tablets, TVs, gaming consoles, 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 2011, there were

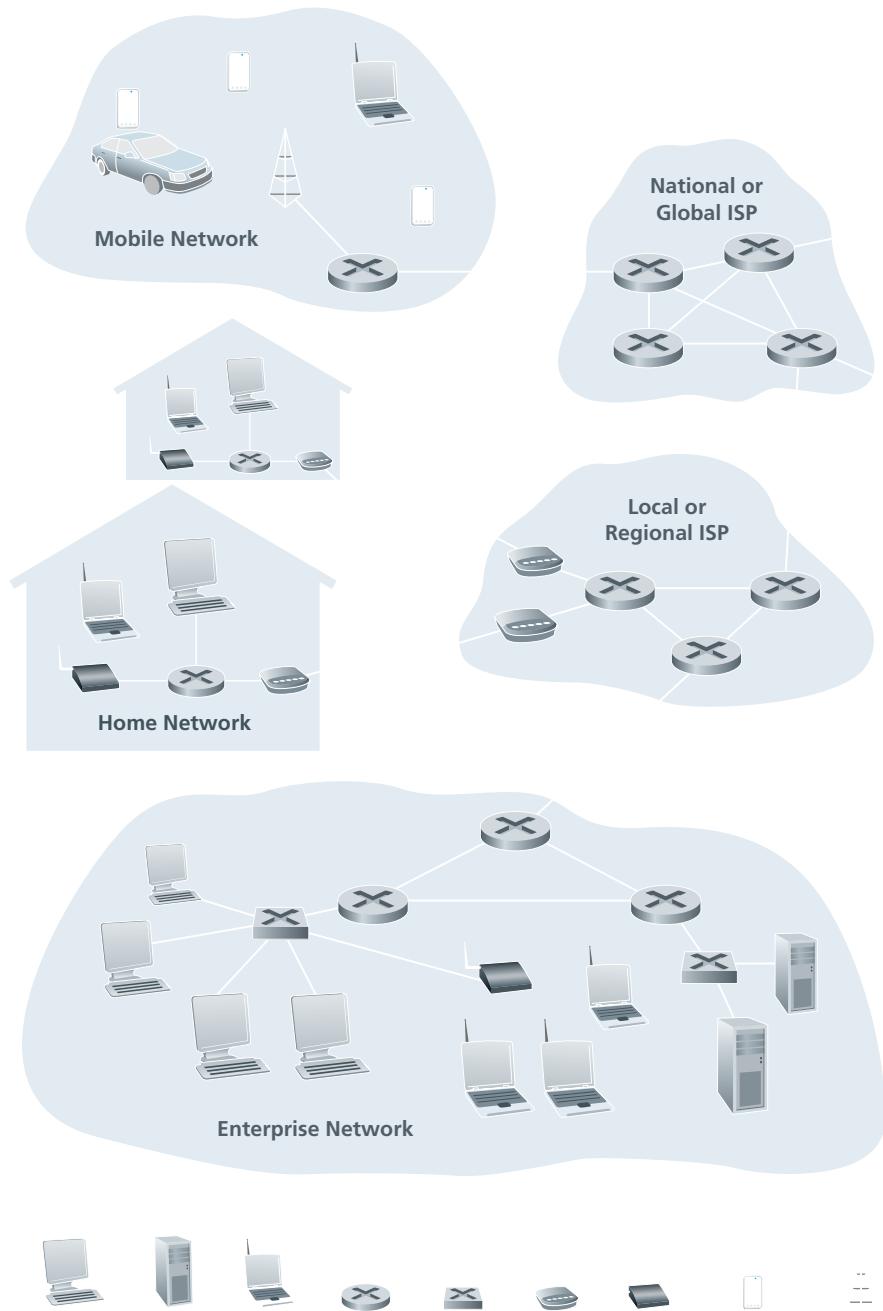


Figure 1.1 ♦ Some pieces of the Internet

nearly 850 million end systems attached to the Internet [ISC 2012], not counting smartphones, laptops, and other devices that are only intermittently connected to the Internet. Overall, more there are an estimated 2 billion Internet users [ITU 2011].

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, optical fiber, 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, known as **packets** in the jargon of computer networks, 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. Link-layer switches are typically used in access networks, while routers are typically used in the network core. The sequence of communication links and packet switches traversed by a packet from the sending end system to the receiving end system is known as a **route** or **path** through the network. The exact amount of traffic being carried in the Internet is difficult to estimate but Cisco [Cisco VNI 2011] estimates global Internet traffic will be nearly 40 exabytes per month in 2012.

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 local cable or telephone companies; corporate ISPs; university ISPs; and ISPs that provide WiFi 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 residential broadband access such as cable modem or DSL,

high-speed local area network access, wireless access, and 56 kbps dial-up modem access. ISPs also provide Internet access to content providers, connecting Web sites directly to the Internet. The Internet is all about connecting end systems to each other, so the ISPs that provide access to end systems must also be interconnected. These lower-tier ISPs are interconnected through national and international upper-tier ISPs such as Level 3 Communications, AT&T, Sprint, and NTT. 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, so that people can create systems and products that interoperate. This is where standards come into play. **Internet standards** are developed by the Internet Engineering Task Force (IETF)[IETF 2012]. 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 [Allman 2011]. 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 currently more than 6,000 RFCs. Other bodies also specify standards for network components, most notably for network links. The IEEE 802 LAN/MAN Standards Committee [IEEE 802 2012], for example, specifies the Ethernet and wireless WiFi standards.

1.1.2 A Services Description

Our discussion above 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, social networks, instant messaging, Voice-over-IP (VoIP), video streaming, distributed games, peer-to-peer (P2P) file sharing, television 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. 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 programs that run on the end systems. You might, for example, write your programs in Java, C, or Python. Now, because you are developing a distributed Internet application, the programs 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 program running on one end system instruct the Internet to deliver data to another program running on another end system?

End systems attached to the Internet provide an **Application Programming Interface (API)** that specifies how a program running on one end system asks the Internet infrastructure to deliver data to a specific destination program running on another end system. This Internet API is a set of rules that the sending program must follow so that the Internet can deliver the data to the destination program. 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 the program sending data must follow to have the Internet deliver the data to the program that will receive the data.

The postal service, of course, provides more than one service to its customers. It provides express delivery, reception confirmation, ordinary use, 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.

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 and TCP/IP? 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?

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.

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

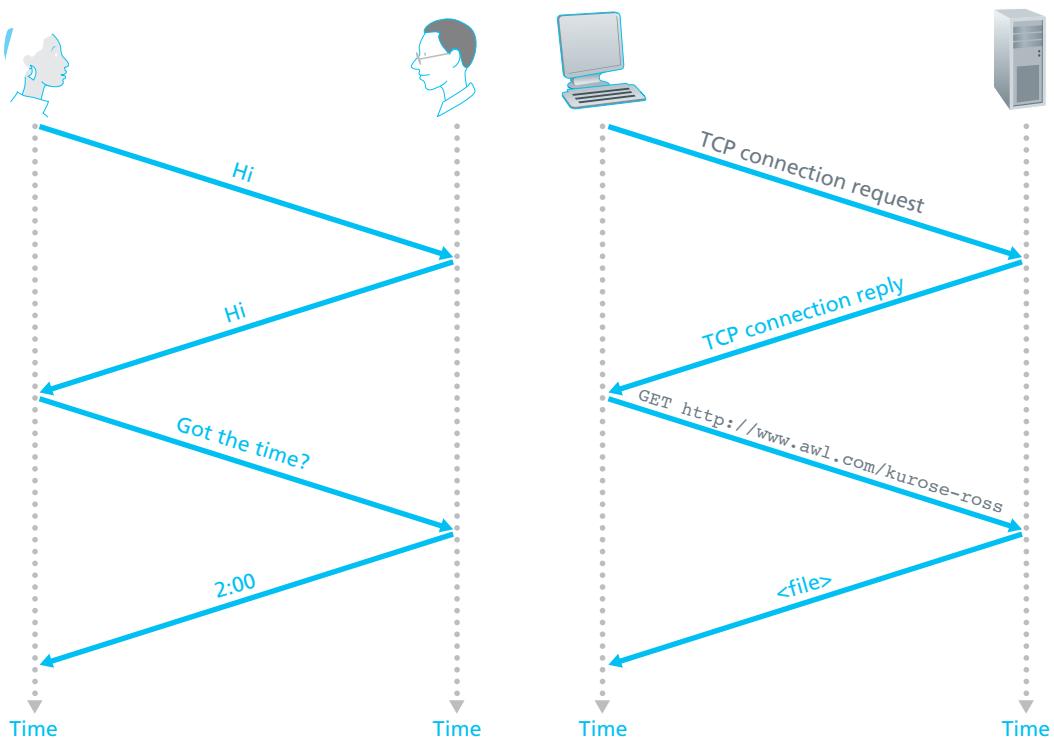


Figure 1.2 ♦ A human protocol and a computer network protocol

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 device (for example, computer, smartphone, tablet, 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 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, smartphones 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.



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 are being connected to the Internet, leveraging their ability to send and receive digital data. 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 network these devices together and to Internet-connected servers.

Many of these devices are based in the home—video game consoles (e.g., Microsoft's Xbox), Internet-ready televisions, digital picture frames that download and display digital pictures, washing machines, refrigerators, and even a toaster that downloads meteorological information and burns an image of the day's forecast (e.g., mixed clouds and sun) on your morning toast [BBC 2001]. IP-enabled phones with GPS capabilities put location-dependent services (maps, information about nearby services or people) at your fingertips. Networked sensors embedded into the physical environment allow monitoring of buildings, bridges, seismic activity, wildlife habitats, river estuaries, and the weather. Biomedical devices can be embedded and networked in a body-area network. With so many diverse devices being networked together, the Internet is indeed becoming an "Internet of things" [ITU 2005b].

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., laptops, smartphones, and tablets). Furthermore, an increasing number of non-traditional 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 client program, or an e-mail server program. Throughout this book we will use the 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, smartphones, 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. Today, most of the servers from which we receive

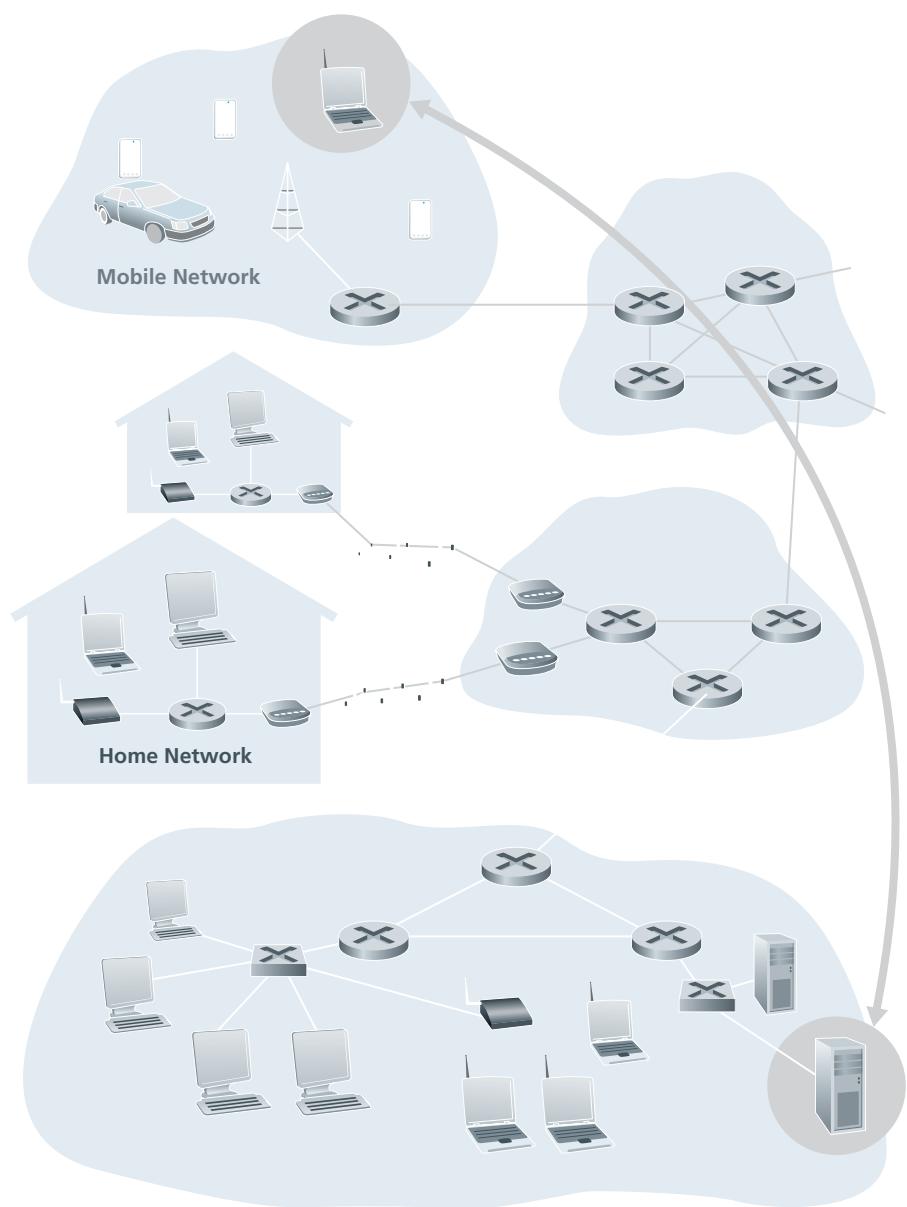


Figure 1.3 ◆ End-system interaction

search results, e-mail, Web pages, and videos reside in large **data centers**. For example, Google has 30–50 data centers, with many having more than one hundred thousand servers.

1.2.1 Access Networks

Having considered the applications and end systems at the “edge of the network,” let’s next consider the access network—the network that physically connects an end system to the first router (also known as the “edge router”) on a path from the end system to any other distant end system. Figure 1.4 shows several types of access

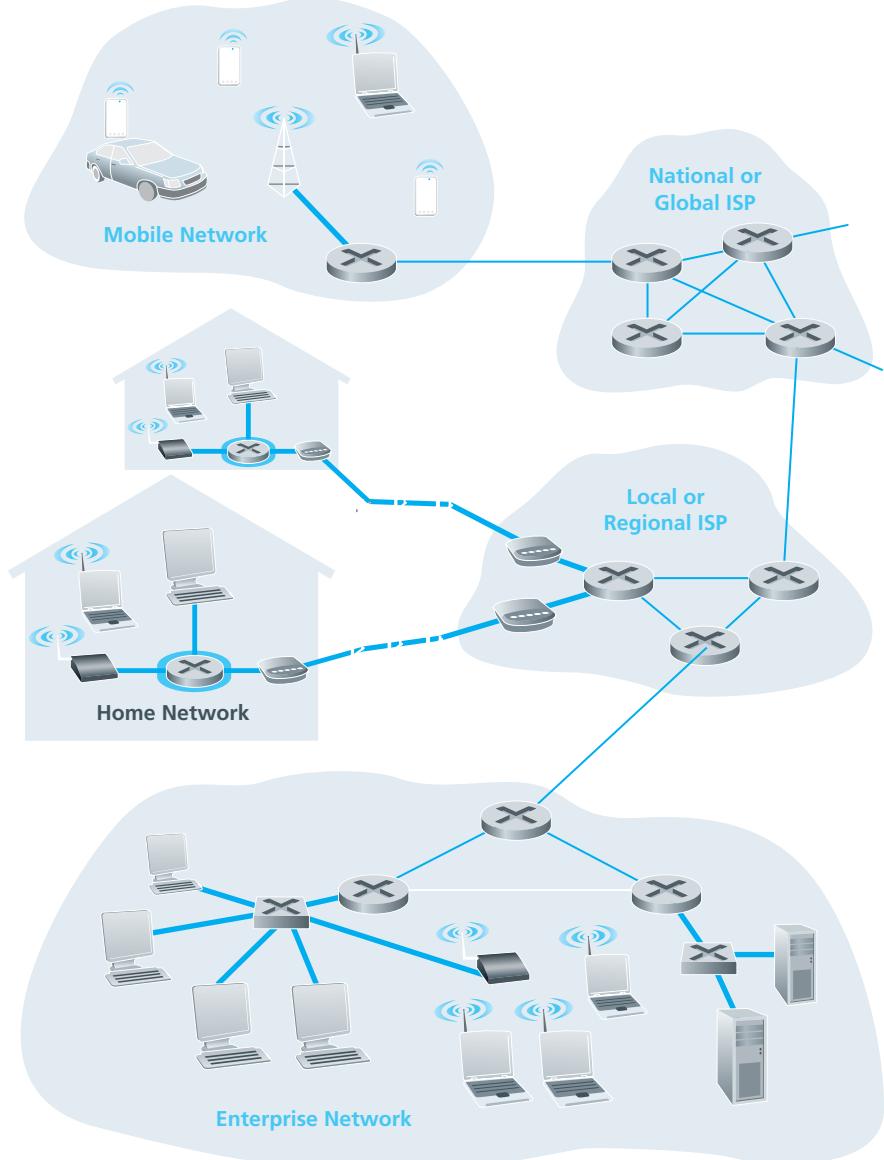


Figure 1.4 ♦ Access networks

networks with thick, shaded lines, and the settings (home, enterprise, and wide-area mobile wireless) in which they are used.

Home Access: DSL, Cable, FTTH, Dial-Up, and Satellite

In developed countries today, more than 65 percent of the households have Internet access, with Korea, Netherlands, Finland, and Sweden leading the way with more than 80 percent of households having Internet access, almost all via a high-speed broadband connection [ITU 2011]. Finland and Spain have recently declared high-speed Internet access to be a “legal right.” Given this intense interest in home access, let’s begin our overview of access networks by considering how homes connect to the Internet.

Today, the two most prevalent types of broadband residential access are **digital subscriber line (DSL)** and cable. A residence typically obtains DSL Internet access from the same local telephone company (telco) that provides its wired local phone access. Thus, when DSL is used, a customer’s telco is also its ISP. As shown in Figure 1.5, each customer’s DSL modem uses the existing telephone line (twisted-pair copper wire, which we’ll discuss in Section 1.2.2) to exchange data with a digital subscriber line access multiplexer (DSLAM) located in the telco’s local central office (CO). The home’s DSL modem takes digital data and translates it to high-frequency tones for transmission over telephone wires to the CO; the analog signals from many such houses are translated back into digital format at the DSLAM.

The residential telephone line carries both data and traditional telephone signals simultaneously, which are encoded at different frequencies:

- A high-speed downstream channel, in the 50 kHz to 1 MHz band
- 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’ll describe this technique of frequency-division multiplexing in

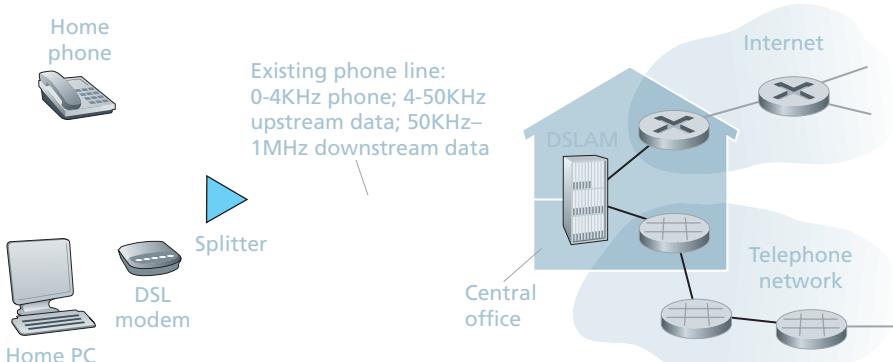


Figure 1.5 ♦ DSL Internet access

Section 1.3.1). On the customer side, a splitter separates the data and telephone signals arriving to the home and forwards the data signal to the DSL modem. On the telco side, in the CO, the DSLAM separates the data and phone signals and sends the data into the Internet. Hundreds or even thousands of households connect to a single DSLAM [Dischinger 2007].

The DSL standards define transmission rates of 12 Mbps downstream and 1.8 Mbps upstream [ITU 1999], and 24 Mbps downstream and 2.5 Mbps upstream [ITU 2003]. Because the downstream and upstream rates are different, the access is said to be asymmetric. The actual downstream and upstream transmission rates achieved may be less than the rates noted above, as the DSL provider may purposefully limit a residential rate when tiered service (different rates, available at different prices) are offered, or because the maximum rate can be limited by the distance between the home and the CO, the gauge of the twisted-pair line and the degree of electrical interference. Engineers have expressly designed DSL for short distances between the home and the CO; generally, if the residence is not located within 5 to 10 miles of the CO, the residence must resort to an alternative form of Internet access.

While DSL makes use of the telco's existing local telephone infrastructure, **cable Internet access** makes use of the cable television company's existing cable television infrastructure. A residence obtains cable Internet access from the same company that provides its cable television. As illustrated in Figure 1.6, fiber optics connect 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. Because both fiber and coaxial cable are employed in this system, it is often referred to as hybrid fiber coax (HFC).

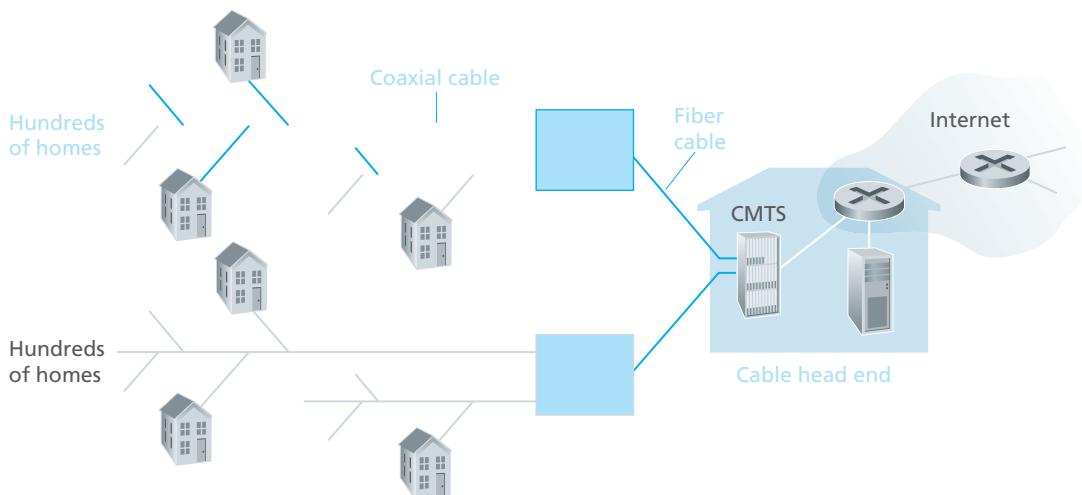


Figure 1.6 ♦ A hybrid fiber-coaxial access network

Cable internet access requires special modems, called cable modems. As with a DSL modem, the cable modem is typically an external device and connects to the home PC through an Ethernet port. (We will discuss Ethernet in great detail in Chapter 5.) At the cable head end, the cable modem termination system (CMTS) serves a similar function as the DSL network's DSLAM—turning the analog signal sent from the cable modems in many downstream homes back into digital format. Cable modems divide the HFC network into two channels, a downstream and an upstream channel. As with DSL, access is typically asymmetric, with the downstream channel typically allocated a higher transmission rate than the upstream channel. The DOCSIS 2.0 standard defines downstream rates up to 42.8 Mbps and upstream rates of up to 30.7 Mbps. As in the case of DSL networks, the maximum achievable rate may not be realized due to lower contracted data rates or media impairments.

One important characteristic of cable Internet access 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 a video file on the downstream channel, the actual rate at which each user receives its video file will be significantly lower than the aggregate cable 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 cable downstream rate, because the 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 and avoid collisions. (We'll discuss this collision issue in some detail in Chapter 5.)

Although DSL and cable networks currently represent more than 90 percent of residential broadband access in the United States, an up-and-coming technology that promises even higher speeds is the deployment of **fiber to the home (FTTH)** [FTTH Council 2011a]. As the name suggests, the FTTH concept is simple—provide an optical fiber path from the CO directly to the home. In the United States, Verizon has been particularly aggressive with FTTH with its FIOS service [Verizon FIOS 2012].

There are several competing technologies for optical distribution from the CO to the homes. The simplest optical distribution network is called direct fiber, with one fiber leaving the CO for each home. More commonly, each fiber leaving the central office is actually shared by many homes; it is not until the fiber gets relatively close to the homes that it is split into individual customer-specific fibers. There are two competing optical-distribution network architectures that perform this splitting: active optical networks (AONs) and passive optical networks (PONs). AON is essentially switched Ethernet, which is discussed in Chapter 5.

Here, we briefly discuss PON, which is used in Verizon's FIOS service. Figure 1.7 shows FTTH using the PON distribution architecture. Each home has

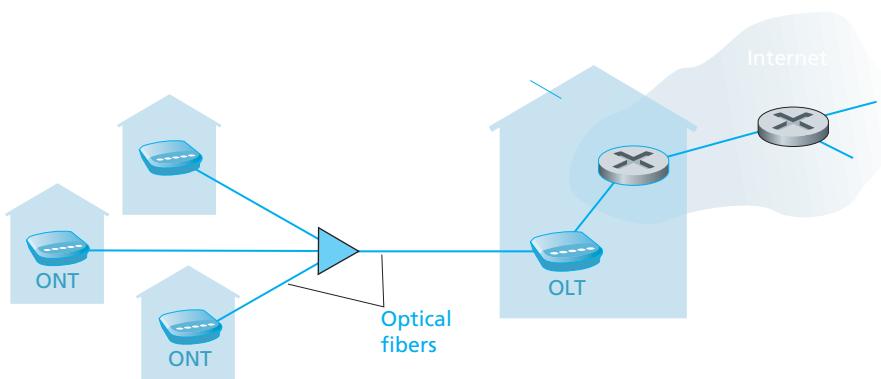


Figure 1.7 ♦ FTTH Internet access

an optical network terminator (ONT), which is connected by dedicated optical fiber to a neighborhood splitter. The splitter combines a number of homes (typically less than 100) onto a single, shared optical fiber, which connects to an optical line terminator (OLT) in the telco's CO. The OLT, providing conversion between optical and electrical signals, connects to the Internet via a telco router. In the home, users connect a home router (typically a wireless router) to the ONT and access the Internet via this home router. In the PON architecture, all packets sent from OLT to the splitter are replicated at the splitter (similar to a cable head end).

FTTH can potentially provide Internet access rates in the gigabits per second range. However, most FTTH ISPs provide different rate offerings, with the higher rates naturally costing more money. The average downstream speed of US FTTH customers was approximately 20 Mbps in 2011 (compared with 13 Mbps for cable access networks and less than 5 Mbps for DSL) [FTTH Council 2011b].

Two other access network technologies are also used to provide Internet access to the home. In locations where DSL, cable, and FTTH are not available (e.g., in some rural settings), 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. Dial-up access over traditional phone lines is based on the same model as DSL—a home modem connects over a phone line to a modem in the ISP. Compared with DSL and other broadband access networks, dial-up access is excruciatingly slow at 56 kbps.

Access in the Enterprise (and the Home): Ethernet and WiFi

On corporate and university campuses, and increasingly in home settings, a local area network (LAN) is used to connect an end system to the edge router. Although there are many types of LAN technologies, Ethernet is by far the most prevalent access technology in corporate, university, and home networks. As shown in Figure 1.8, Ethernet users use twisted-pair copper wire to connect to an Ethernet switch, a

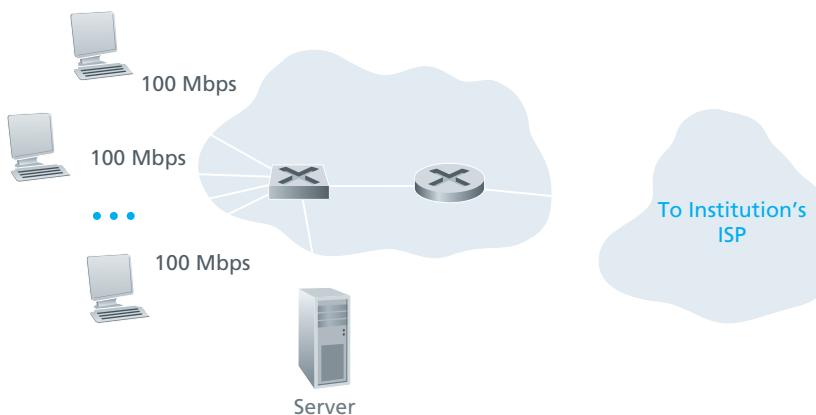


Figure 1.8 ♦ Ethernet Internet access

technology discussed in detail in Chapter 5. The Ethernet switch, or a network of such interconnected switches, is then in turn connected into the larger Internet. With Ethernet access, users typically have 100 Mbps access to the Ethernet switch, whereas servers may have 1 Gbps or even 10 Gbps access.

Increasingly, however, people are accessing the Internet wirelessly from laptops, smartphones, tablets, and other devices (see earlier sidebar on “A Dizzying Array of Devices”). In a wireless LAN setting, wireless users transmit/receive packets to/from an access point that is connected into the enterprise’s network (most likely including wired Ethernet), which in turn is connected to the wired Internet. A wireless LAN user must typically be within a few tens of meters of the access point. Wireless LAN access based on IEEE 802.11 technology, more colloquially known as WiFi, is now just about everywhere—universities, business offices, cafes, airports, homes, and even in airplanes. 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 [wigle.net 2012]). As discussed in detail in Chapter 6, 802.11 today provides a shared transmission rate of up to 54 Mbps.

Even though Ethernet and WiFi access networks were initially deployed in enterprise (corporate, university) settings, they have recently become relatively common components of home networks. Many homes combine broadband residential access (that is, cable modems or DSL) with these inexpensive wireless LAN technologies to create powerful home networks [Edwards 2011]. Figure 1.9 shows 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 the kitchen to the backyard to the bedrooms.

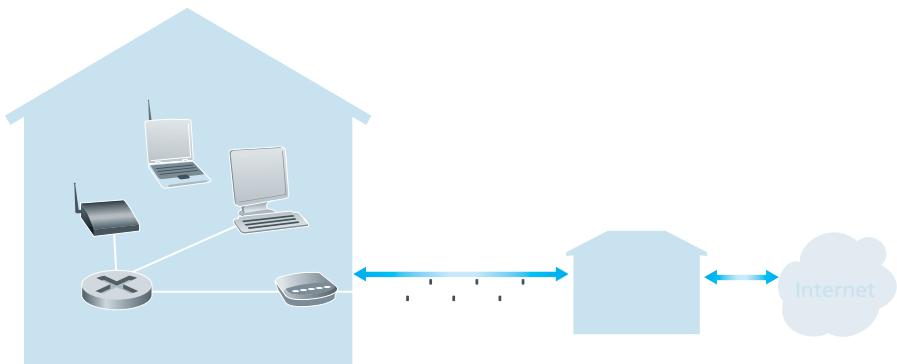


Figure 1.9 ♦ A typical home network

Wide-Area Wireless Access: 3G and LTE

Increasingly, devices such as iPhones, BlackBerrys, and Android devices are being used to send email, surf the Web, Tweet, and download music while on the run. These devices employ the same wireless infrastructure used for cellular telephony to send/receive packets through a base station that is operated by the cellular network provider. Unlike WiFi, a user need only be within a few tens of kilometers (as opposed to a few tens of meters) of the base station.

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. But even higher-speed wide-area access technologies—a fourth-generation (4G) of wide-area wireless networks—are already being deployed. LTE (for “Long-Term Evolution”—a candidate for Bad Acronym of the Year Award) has its roots in 3G technology, and can potentially achieve rates in excess of 10 Mbps. LTE downstream rates of many tens of Mbps have been reported in commercial deployments. We’ll cover the basic principles of wireless networks and mobility, as well as WiFi, 3G, and LTE technologies (and more!) in Chapter 6.

1.2.2 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 DSL and Ethernet use 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. 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 10 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 6a cable, can achieve data rates of 10 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 tens of 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 achieve high data transmission 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 tens of Mbps. 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, 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-48, OC-96, OC-192, OC-768. [Mukherjee 2006, Ramaswamy 2010] 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 three groups: those that operate over very short distance (e.g., with one or two meters); 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. Personal devices such as wireless headsets, keyboards, and medical devices operate over short distances; the wireless LAN technologies described in Section 1.2.1 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 areas without access to DSL or cable-based Internet access.

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 2012] 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 1.10 highlights the network core with thick, shaded lines.

1.3.1 Packet Switching

In a network application, end systems exchange **messages** with each other. Messages can contain anything the application designer wants. Messages may perform a control function (for example, the “Hi” messages in our handshaking example in Figure 1.2) or can contain data, such as an email message, a JPEG image, or an MP3 audio file. To send a message from a source end system to a destination end system, the source breaks long messages into smaller chunks of data known as **packets**. Between source and destination, each packet 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. So, if a source end system or a packet switch is sending a packet of L bits over a link with transmission rate R bits/sec, then the time to transmit the packet is L/R seconds.

Store-and-Forward Transmission

Most packet switches use **store-and-forward transmission** at the inputs to the links. Store-and-forward transmission means that the packet switch must receive the entire packet before it can begin to transmit the first bit of the packet onto the outbound link. To explore store-and-forward transmission in more detail, consider a simple network consisting of two end systems connected by a single router, as shown in Figure 1.11. A router will typically have many incident links, since its job is to switch an incoming packet onto an outgoing link; in this simple example, the router has the rather simple task of transferring a packet from one (input) link to the only other attached link. In this example, the source has three packets, each consisting of L bits, to send to the destination. At the snapshot of time shown in Figure 1.11, the source has transmitted some of packet 1, and the front of packet 1 has already arrived at the router. Because the router employs store-and-forwarding, at this instant of time, the router cannot transmit the bits it has received; instead it

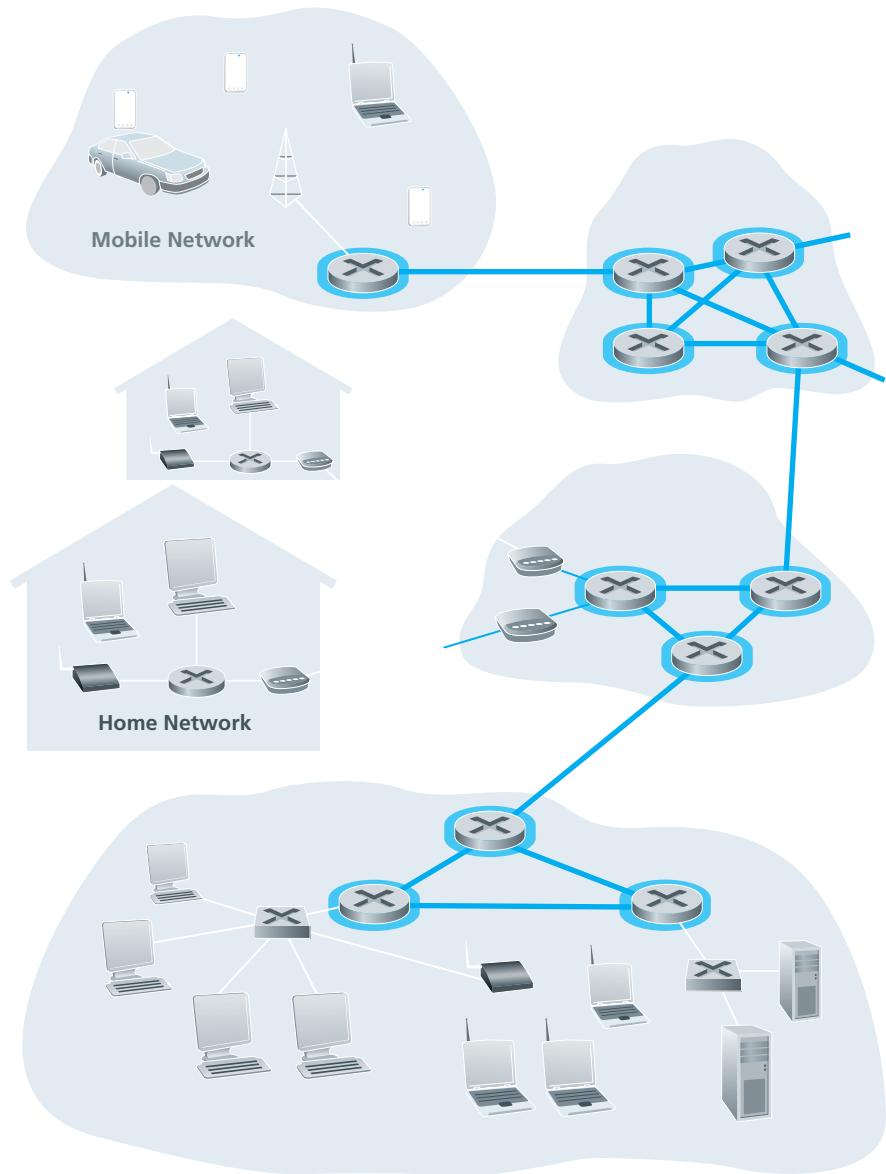


Figure 1.10 ♦ The network core

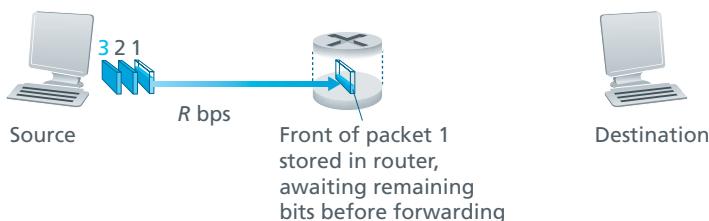


Figure 1.11 ◆ Store-and-forward packet switching

must first buffer (i.e., “store”) the packet’s bits. Only after the router has received *all* of the packet’s bits can it begin to transmit (i.e., “forward”) the packet onto the outbound link. To gain some insight into store-and-forward transmission, let’s now calculate the amount of time that elapses from when the source begins to send the packet until the destination has received the entire packet. (Here we will ignore propagation delay—the time it takes for the bits to travel across the wire at near the speed of light—which will be discussed in Section 1.4.) The source begins to transmit at time 0; at time L/R seconds, the source has transmitted the entire packet, and the entire packet has been received and stored at the router (since there is no propagation delay). At time L/R seconds, since the router has just received the entire packet, it can begin to transmit the packet onto the outbound link towards the destination; at time $2L/R$, the router has transmitted the entire packet, and the entire packet has been received by the destination. Thus, the total delay is $2L/R$. If the switch instead forwarded bits as soon as they arrive (without first receiving the entire packet), then the total delay would be L/R since bits are not held up at the router. But, as we will discuss in Section 1.4, routers need to receive, store, and *process* the entire packet before forwarding.

Now let’s calculate the amount of time that elapses from when the source begins to send the first packet until the destination has received all three packets. As before, at time L/R , the router begins to forward the first packet. But also at time L/R the source will begin to send the second packet, since it has just finished sending the entire first packet. Thus, at time $2L/R$, the destination has received the first packet and the router has received the second packet. Similarly, at time $3L/R$, the destination has received the first two packets and the router has received the third packet. Finally, at time $4L/R$ the destination has received all three packets!

Let’s now consider the general case of sending one packet from source to destination over a path consisting of N links each of rate R (thus, there are $N-1$ routers between source and destination). Applying the same logic as above, we see that the end-to-end delay is:

$$d_{\text{end-to-end}} = N \frac{L}{R} \quad (1.1)$$

You may now want to try to determine what the delay would be for P packets sent over a series of N links.

Queuing Delays and Packet Loss

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 onto 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 full 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.

Figure 1.12 illustrates a simple packet-switched network. As in Figure 1.11, 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 router. The router then directs these packets to the 1.5 Mbps link. If, during a short interval of time, the arrival rate of packets to the router (when converted to bits per second) exceeds 1.5 Mbps, congestion will occur at the router as packets queue in the link's output buffer before being transmitted onto the link. For example, if Host A and B each send a burst of five packets back-to-back at the same time, then most of these packets will spend some time waiting in the queue. The situation is, in fact, entirely analogous to many common-day situations—for example, when we wait in line for a bank teller or wait in front of a tollbooth. We'll examine this queuing delay in more detail in Section 1.4.

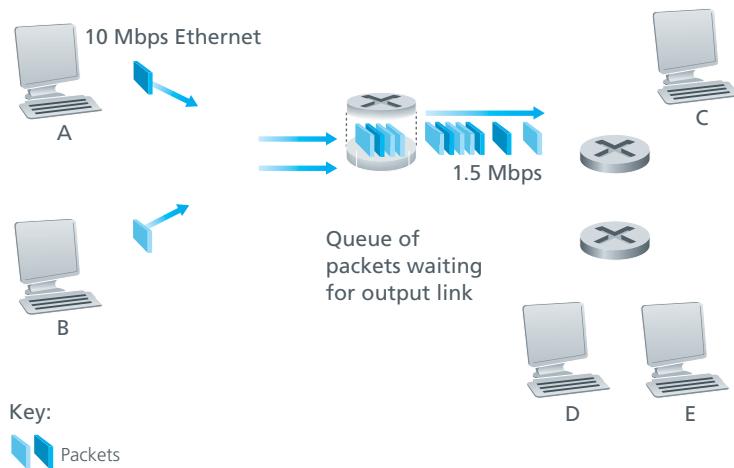


Figure 1.12 ◆ Packet switching

Forwarding Tables and Routing Protocols

Earlier, we said that a router takes a packet arriving on one of its attached communication links and forwards that packet onto another one of its attached communication links. But how does the router determine which link it should forward the packet onto? Packet forwarding is actually done in different ways in different types of computer networks. Here, we briefly describe how it is done in the Internet.

In the Internet, every end system has an address called an IP address. When a source end system wants to send a packet to a destination end system, the source includes the destination's IP address in the packet's 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 that router's outbound links. When a packet arrives at a router, the router examines the address and searches its forwarding table, using this destination address, to find the appropriate outbound link. The router then directs the packet to this outbound link.

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 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.

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.

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. Simply visit the site www.traceroute.org, choose a source in a particular country, and trace the route from that source to your computer. (For a discussion of Traceroute, see Section 1.4.)

1.3.2 Circuit Switching

There are two fundamental approaches to moving data through a network of links and switches: **circuit switching** and **packet switching**. Having covered packet-switched networks in the previous subsection, we now turn our attention to circuit-switched networks.

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 be seated 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 be seated.

Traditional 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 (representing a fraction of each link's transmission capacity) for the duration of the connection. Since a given transmission rate has been reserved for this sender-to-receiver connection, the sender can transfer the data to the receiver at the *guaranteed* constant rate.

Figure 1.13 illustrates a circuit-switched network. In this network, the four circuit switches are interconnected by four links. Each of these links has four circuits, so that each link can support four 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. Thus, in order for Host A to communicate with Host B, the network must first reserve one circuit on each of two links. In this example, the dedicated end-to-end connection uses the second circuit in the first link and the fourth circuit in the second link. Because each link has four circuits, for each link used by the end-to-end connection, the connection gets one fourth of the link's total transmission capacity for the duration of the connection. Thus, for example, if each link between adjacent switches has a transmission rate of 1 Mbps, then each end-to-end circuit-switch connection gets 250 kbps of dedicated transmission rate.

In contrast, consider what happens when one host wants to send a packet to another host over a packet-switched network, such as the Internet. As with circuit switching, the packet is transmitted over a series of communication links. But different from circuit switching, the packet is sent into the network without reserving any link resources whatsoever. If one of the links is congested because other packets need to be transmitted over the link at the same time, then the 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.

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.

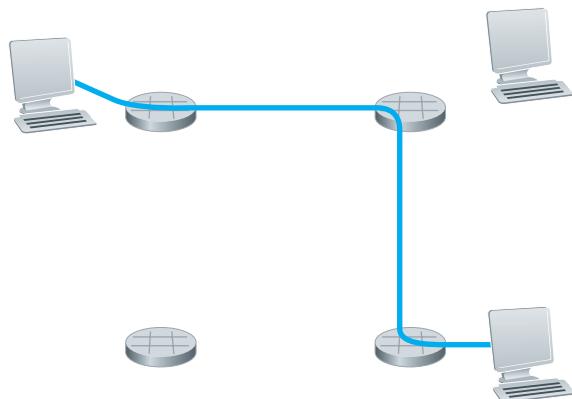


Figure 1.13 ♦ A simple circuit-switched network consisting of four switches and four links

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 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 1.14 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,

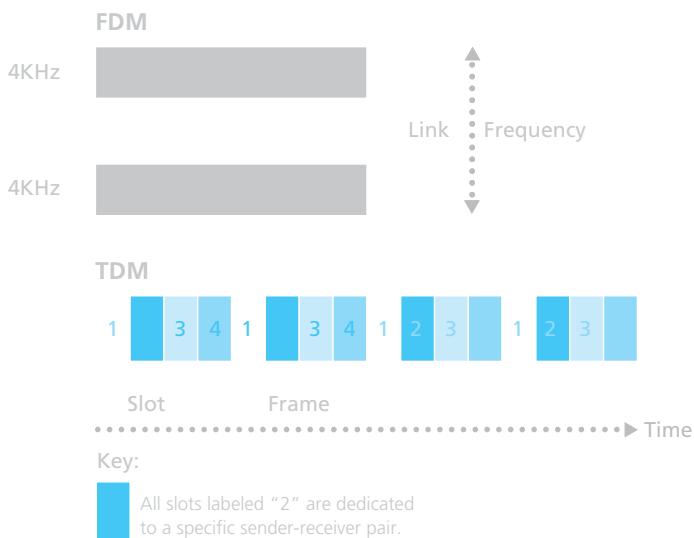


Figure 1.14 ♦ With FDM, each circuit continuously gets a fraction of the bandwidth. With TDM, each circuit gets all of the bandwidth periodically during brief intervals of time (that is, during slots)

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 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 transmission capacity 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 $(1.536 \text{ Mbps})/24 = 64 \text{ kbps}$, so it takes $(640,000 \text{ bits})/(64 \text{ kbps}) = 10 \text{ seconds}$ 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.)

Packet Switching Versus Circuit Switching

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 transmission capacity 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 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 ($= 1 \text{ Mbps}/100 \text{ kbps}$) 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 P8 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 time 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.

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.

1.3.3 A Network of Networks

We saw earlier that end systems (PCs, smartphones, Web servers, mail servers, and so on) connect into the Internet via an access ISP. The access ISP can provide either wired or wireless connectivity, using an array of access technologies including DSL, cable, FTTH, Wi-Fi, and cellular. Note that the access ISP does not have to be a telco or a cable company; instead it can be, for example, a university (providing Internet access to students, staff, and faculty), or a company (providing access for its employees). But connecting end users and content providers into an access ISP is only a small piece of solving the puzzle of connecting the billions of end systems that make up the Internet. To complete this puzzle, the access ISPs themselves must be interconnected. This is done by creating a *network of networks*—understanding this phrase is the key to understanding the Internet.

Over the years, the network of networks that forms the Internet has evolved into a very complex structure. Much of this evolution is driven by economics and national policy, rather than by performance considerations. In order to understand today's Internet network structure, let's incrementally build a series of network structures, with each new structure being a better approximation of the complex Internet that we have today. Recall that the overarching goal is to interconnect the access ISPs so that all end systems can send packets to each other. One naive approach would be to have each access ISP *directly* connect with every other access ISP. Such a mesh design is, of course, much too costly for the access ISPs, as it would require each access ISP to have a separate communication link to each of the hundreds of thousands of other access ISPs all over the world.

Our first network structure, *Network Structure 1*, interconnects all of the access ISPs with a *single global transit ISP*. Our (imaginary) global transit ISP is a network of routers and communication links that not only spans the globe, but also has at least one router near each of the hundreds of thousands of access ISPs. Of course, it would be very costly for the global ISP to build such an extensive network. To be profitable, it would naturally charge each of the access ISPs for connectivity, with the pricing reflecting (but not necessarily directly proportional to) the amount of traffic an access ISP exchanges with the global ISP. Since the access ISP pays the global transit ISP, the access ISP is said to be a **customer** and the global transit ISP is said to be a **provider**.

Now if some company builds and operates a global transit ISP that is profitable, then it is natural for other companies to build their own global transit ISPs and compete with the original global transit ISP. This leads to *Network Structure 2*, which consists of the hundreds of thousands of access ISPs and *multiple* global transit ISPs. The access ISPs certainly prefer Network Structure 2 over Network Structure 1 since they can now choose among the competing global transit providers as a function of their pricing and services. Note, however, that the global transit ISPs

themselves must interconnect: Otherwise access ISPs connected to one of the global transit providers would not be able to communicate with access ISPs connected to the other global transit providers.

Network Structure 2, just described, is a two-tier hierarchy with global transit providers residing at the top tier and access ISPs at the bottom tier. This assumes that global transit ISPs are not only capable of getting close to each and every access ISP, but also find it economically desirable to do so. In reality, although some ISPs do have impressive global coverage and do directly connect with many access ISPs, no ISP has presence in each and every city in the world. Instead, in any given region, there may be a **regional ISP** to which the access ISPs in the region connect. Each regional ISP then connects to **tier-1 ISPs**. Tier-1 ISPs are similar to our (imaginary) global transit ISP; but tier-1 ISPs, which actually do exist, do not have a presence in every city in the world. There are approximately a dozen tier-1 ISPs, including Level 3 Communications, AT&T, Sprint, and NTT. Interestingly, no group officially sanctions tier-1 status; as the saying goes—if you have to ask if you’re a member of a group, you’re probably not.

Returning to this network of networks, not only are there multiple competing tier-1 ISPs, there may be multiple competing regional ISPs in a region. In such a hierarchy, each access ISP pays the regional ISP to which it connects, and each regional ISP pays the tier-1 ISP to which it connects. (An access ISP can also connect directly to a tier-1 ISP, in which case it pays the tier-1 ISP). Thus, there is customer-provider relationship at each level of the hierarchy. Note that the tier-1 ISPs do not pay anyone as they are at the top of the hierarchy. To further complicate matters, in some regions, there may be a larger regional ISP (possibly spanning an entire country) to which the smaller regional ISPs in that region connect; the larger regional ISP then connects to a tier-1 ISP. For example, in China, there are access ISPs in each city, which connect to provincial ISPs, which in turn connect to national ISPs, which finally connect to tier-1 ISPs [Tian 2012]. We refer to this multi-tier hierarchy, which is still only a crude approximation of today’s Internet, as *Network Structure 3*.

To build a network that more closely resembles today’s Internet, we must add points of presence (PoPs), multi-homing, peering, and Internet exchange points (IXPs) to the hierarchical Network Structure 3. PoPs exist in all levels of the hierarchy, except for the bottom (access ISP) level. A **PoP** is simply a group of one or more routers (at the same location) in the provider’s network where customer ISPs can connect into the provider ISP. For a customer network to connect to a provider’s PoP, it can lease a high-speed link from a third-party telecommunications provider to directly connect one of its routers to a router at the PoP. Any ISP (except for tier-1 ISPs) may choose to **multi-home**, that is, to connect to two or more provider ISPs. So, for example, an access ISP may multi-home with two regional ISPs, or it may multi-home with two regional ISPs and also with a tier-1 ISP. Similarly, a regional ISP may multi-home with multiple tier-1 ISPs. When an

ISP multi-homes, it can continue to send and receive packets into the Internet even if one of its providers has a failure.

As we just learned, customer ISPs pay their provider ISPs to obtain global Internet interconnectivity. The amount that a customer ISP pays a provider ISP reflects the amount of traffic it exchanges with the provider. To reduce these costs, a pair of nearby ISPs at the same level of the hierarchy can **peer**, that is, they can directly connect their networks together so that all the traffic between them passes over the direct connection rather than through upstream intermediaries. When two ISPs peer, it is typically settlement-free, that is, neither ISP pays the other. As noted earlier, tier-1 ISPs also peer with one another, settlement-free. For a readable discussion of peering and customer-provider relationships, see [Van der Berg 2008]. Along these same lines, a third-party company can create an **Internet Exchange Point (IXP)** (typically in a stand-alone building with its own switches), which is a meeting point where multiple ISPs can peer together. There are roughly 300 IXPs in the Internet today [Augustin 2009]. We refer to this ecosystem—consisting of access ISPs, regional ISPs, tier-1 ISPs, PoPs, multi-homing, peering, and IXPs—as *Network Structure 4*.

We now finally arrive at *Network Structure 5*, which describes the Internet of 2012. Network Structure 5, illustrated in Figure 1.15, builds on top of Network Structure 4 by adding **content provider networks**. Google is currently one of the leading examples of such a content provider network. As of this writing, it is estimated that Google has 30 to 50 data centers distributed across North America, Europe, Asia, South America, and Australia. Some of these data centers house over one hundred thousand servers, while other data centers are smaller, housing only hundreds of servers. The Google data centers are all interconnected via Google’s private TCP/IP network, which spans the entire globe but is nevertheless separate from the public Internet. Importantly, the Google private network only

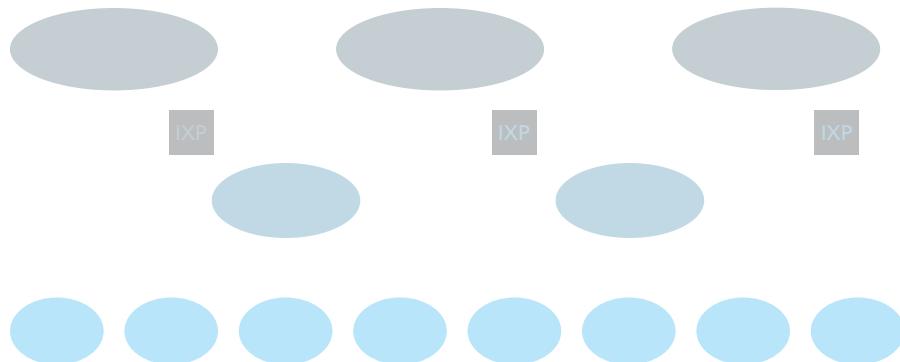


Figure 1.15 ♦ Interconnection of ISPs

carries traffic to/from Google servers. As shown in Figure 1.15, the Google private network attempts to “bypass” the upper tiers of the Internet by peering (settlement free) with lower-tier ISPs, either by directly connecting with them or by connecting with them at IXPs [Labovitz 2010]. However, because many access ISPs can still only be reached by transiting through tier-1 networks, the Google network also connects to tier-1 ISPs, and pays those ISPs for the traffic it exchanges with them. By creating its own network, a content provider not only reduces its payments to upper-tier ISPs, but also has greater control of how its services are ultimately delivered to end users. Google’s network infrastructure is described in greater detail in Section 7.2.4.

In summary, today’s Internet—a network of networks—is complex, consisting of a dozen or so tier-1 ISPs and hundreds of thousands of lower-tier ISPs. The ISPs are diverse in their coverage, with some spanning multiple continents and oceans, and others limited to narrow geographic regions. 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. In recent years, major content providers have also created their own networks and connect directly into lower-tier ISPs where possible.

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 loss of data. 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 thousands 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

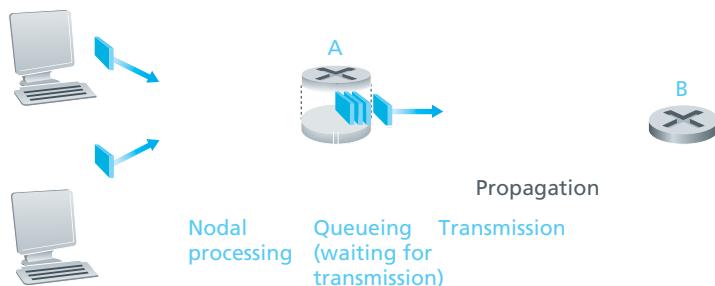


Figure 1.16 ♦ The nodal delay at router A

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**. The performance of many Internet applications—such as search, Web browsing, email, maps, instant messaging, and voice-over-IP—are greatly affected by network delays. 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 1.16. 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 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 onto 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** 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

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

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

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 1.17. 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 $(10 \text{ cars})/(5 \text{ cars/minute}) = 2 \text{ minutes}$. 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 $100 \text{ km}/(100 \text{ km/hour}) = 1 \text{ hour}$. 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.

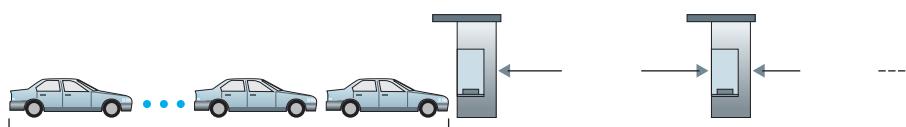


Figure 1.17 ♦ Caravan analogy

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 toll-booth 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. [Smith 2009] also provides a very readable discussion of propagation, queueing, and transmission delays.

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

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

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

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 n th 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 queuing 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 1.18.

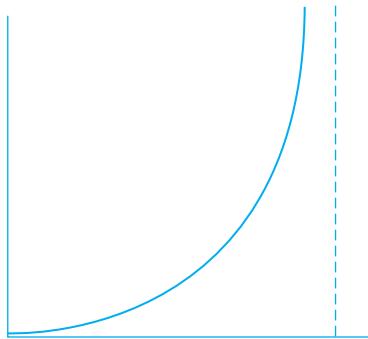


Figure 1.18 ♦ Dependence of average queuing delay on traffic intensity

One important aspect of Figure 1.18 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 router 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

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,

$$d_{\text{end-end}} = N(d_{\text{proc}} + d_{\text{trans}} + d_{\text{prop}}) \quad (1.2)$$

where, once again, $d_{\text{trans}} = L/R$, where L is the packet size. Note that Equation 1.2 is a generalization of Equation 1.1, which did not take into account processing and propagation delays. We leave it to you to generalize Equation 1.2 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 n th router receives the n th 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 N th 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.

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 phenomenon 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 favorites is PingPlotter [PingPlotter 2012].

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, an end system wanting to

transmit a packet into a shared medium (e.g., as in a WiFi or cable modem 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 1.19(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 it receives them. In this case, bits will only leave the router at rate R_c , giving an

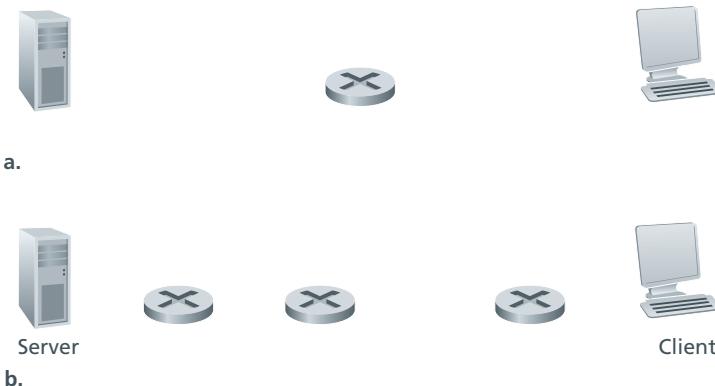


Figure 1.19 ♦ Throughput for a file transfer from server to client

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 store-and-forward and processing delays as well as protocol issues.

Figure 1.19(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, \dots, 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 $\min\{R_1, R_2, \dots, R_N\}$, 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 1.20(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. 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 bits can flow

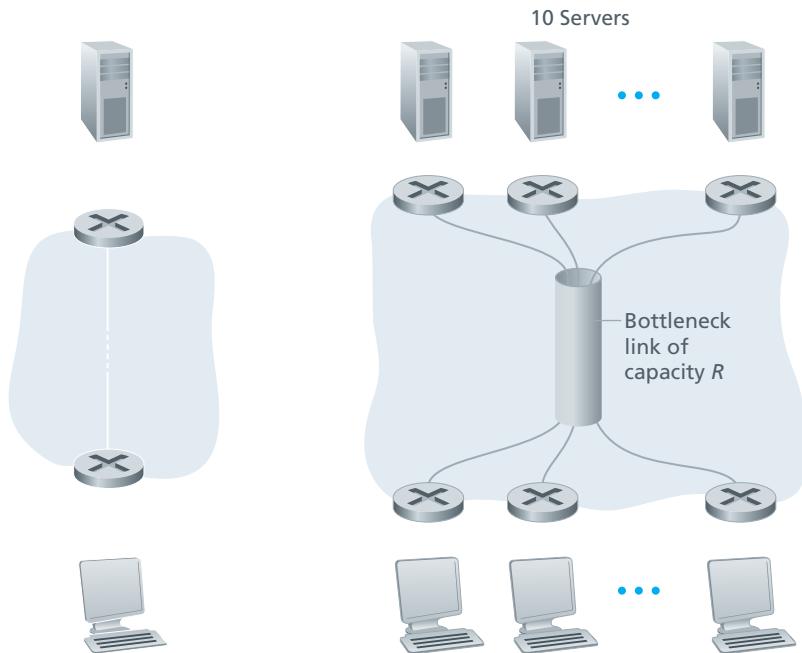


Figure 1.20 ◆ End-to-end throughput: (a) Client downloads a file from server; (b) 10 clients downloading with 10 servers

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 typically the access network.

For a final example, consider Figure 1.20(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

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 1.19 and Figure 1.20(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 1.20(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 1.21.

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

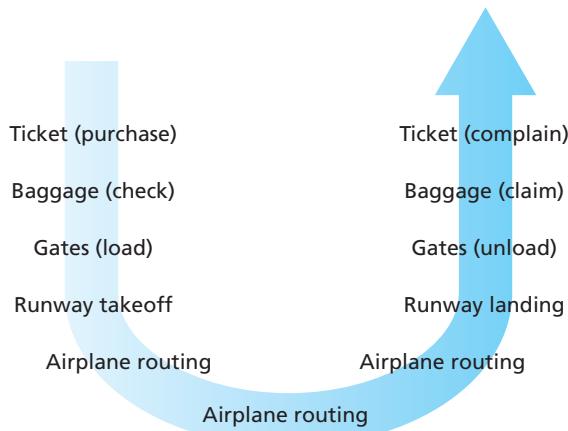


Figure 1.21 ♦ Taking an airplane trip: actions

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 1.21. Looking at Figure 1.21, 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 1.21 in a *horizontal* manner, as shown in Figure 1.22.

Figure 1.22 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 1.22 ♦ Horizontal layering of airline functionality

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 1.22 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

Figure 1.23 ♦ 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 [RFC 3439]. 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 1.23(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.

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

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 cable access network's DOCSIS protocol. 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

organized around seven layers, called the Open Systems Interconnection (OSI) model [ISO 2012]. 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 1.23(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 Encapsulation

Figure 1.24 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 1.24, 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

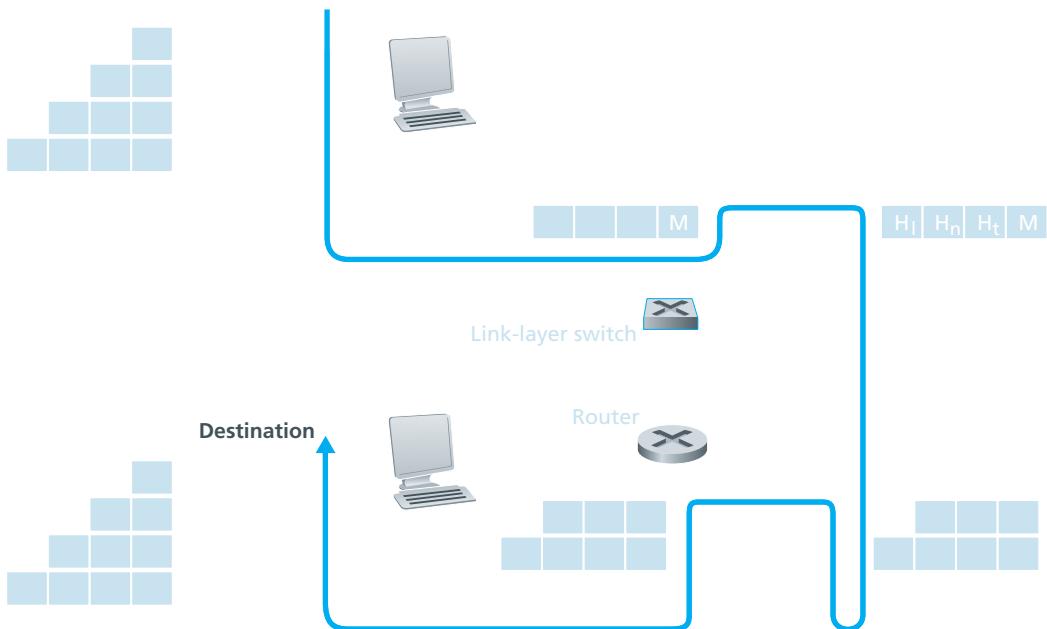


Figure 1.24 ◆ Hosts, routers, and link-layer switches; each contains a different set of layers, reflecting their differences in functionality

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 1.24 also illustrates the important concept of **encapsulation**. At the sending host, an **application-layer message** (M in Figure 1.24) 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 1.24) 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 1.24) 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 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.

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. One of the features of this textbook is that it brings network security issues to the forefront.

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. Malware can spread in the form of a virus or a worm. **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** are malware that can enter a device without any explicit user interaction. For example, a user may be 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. Today, malware, is pervasive and costly to defend against. 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

Another broad class of security threats are known 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 1.25, the attacker controls multiple sources 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

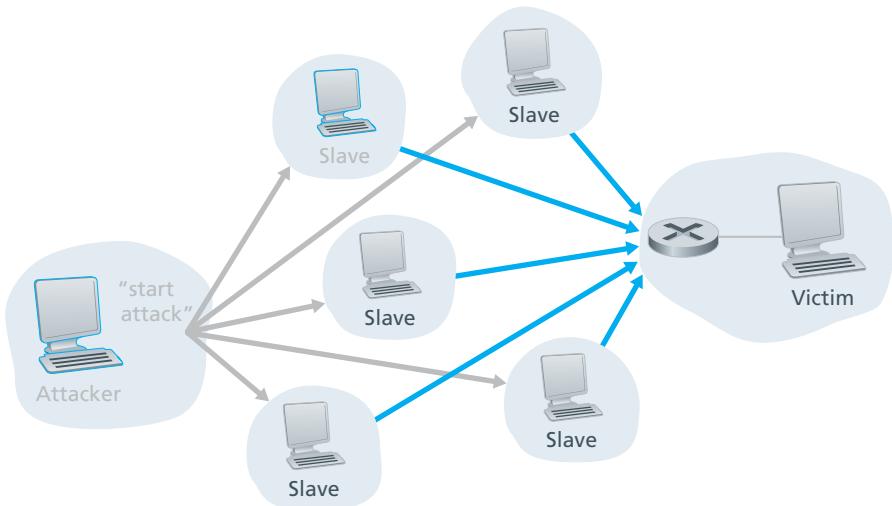


Figure 1.25 ♦ A distributed denial-of-service attack

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 broadcast 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 Wireshark [Wireshark 2012] labs associated with this text (see the introductory Wireshark 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 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.

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 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 of computers in the early 1960s and the advent of timeshared computers, it was perhaps natural to consider 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. On Labor Day in 1969, the first packet switch was installed at UCLA under Kleinrock's supervision, and three additional packet switches were installed shortly thereafter at the Stanford Research Institute (SRI), UC Santa Barbara, and the University of Utah (Figure 1.26). The fledgling precursor to the



Figure 1.26 ♦ An early packet switch

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. 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 wrote the first e-mail program in 1972.

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]; Telenet, a BBN commercial packet-switching network based on ARPAnet technology; Cyclades, a French packet-switching network pioneered by Louis Pouzin [Think 2012]; 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. 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.

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 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. 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. 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 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, including search (e.g., Google and Bing) Internet commerce (e.g., Amazon and eBay) and social networks (e.g., Facebook).

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 2012]. 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 1993 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 along with Jim Clark, formed Mosaic Communications, which later became Netscape Communications Corporation [Cusumano 1998; Quittner 1998]. By 1995, university students were using 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 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. 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
- 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 The New Millennium

Innovation in computer networking continues at a rapid pace. Advances are being made on all fronts, including deployments of faster routers and higher transmission speeds in both access networks and in network backbones. But the following developments merit special attention:

- Since the beginning of the millennium, we have been seeing aggressive deployment of broadband Internet access to homes—not only cable modems and DSL but also fiber to the home, as discussed in Section 1.2. This high-speed Internet access has set the stage for a wealth of video applications, including the distribution of user-generated video (for example, YouTube), on-demand streaming of movies and television shows (e.g., Netflix) , and multi-person video conference (e.g., Skype).
- The increasing ubiquity of high-speed (54 Mbps and higher) public WiFi networks and medium-speed (up to a few Mbps) Internet access via 3G and 4G cellular telephony networks is not only making it possible to remain constantly connected while on the move, but also enabling new location-specific applications. The number of wireless devices connecting to the Internet surpassed the number of wired devices in 2011. This high-speed wireless access has set the stage for the rapid emergence of hand-held computers (iPhones, Androids, iPads, and so on), which enjoy constant and untethered access to the Internet.
- Online social networks, such as Facebook and Twitter, have created massive people networks on top of the Internet. Many Internet users today “live” primarily within Facebook. Through their APIs, the online social networks create platforms for new networked applications and distributed games.

- As discussed in Section 1.3.3, online service providers, such as Google and Microsoft, have deployed their own extensive private networks, which not only connect together their globally distributed data centers, but are used to bypass the Internet as much as possible by peering directly with lower-tier ISPs. As a result, Google provides search results and email access almost instantaneously, as if their data centers were running within one’s own computer.
- Many Internet commerce companies are now running their applications in the “cloud”—such as in Amazon’s EC2, in Google’s Application Engine, or in Microsoft’s Azure. Many companies and universities have also migrated their Internet applications (e.g., email and Web hosting) to the cloud. Cloud companies not only provide applications scalable computing and storage environments, but also provide the applications implicit access to their high-performance private networks.

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 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 Mobile Networks
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 mobile networks, including wireless LANs (including WiFi and Bluetooth), Cellular telephony networks (including GSM, 3G, and 4G), 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.



Homework Problems and Questions

Chapter 1 Review Questions

SECTION 1.1

- R1. What is the difference between a host and an end system? List several different types of end systems. Is a Web server an end system?
- R2. The word *protocol* is often used to describe diplomatic relations. How does Wikipedia describe diplomatic protocol?
- R3. Why are standards important for protocols?

SECTION 1.2

- R4. List six access technologies. Classify each one as home access, enterprise access, or wide-area wireless access.
- R5. Is HFC transmission rate dedicated or shared among users? Are collisions possible in a downstream HFC channel? Why or why not?
- R6. List the available residential access technologies in your city. For each type of access, provide the advertised downstream rate, upstream rate, and monthly price.
- R7. What is the transmission rate of Ethernet LANs?
- R8. What are some of the physical media that Ethernet can run over?
- R9. Dial-up modems, HFC, DSL and FTTH are all used for residential access. For each of these access technologies, provide a range of transmission rates and comment on whether the transmission rate is shared or dedicated.
- R10. Describe the most popular wireless Internet access technologies today. Compare and contrast them.

SECTION 1.3

- R11. Suppose there is exactly one packet switch between a sending host and a receiving host. The transmission rates between the sending host and the switch and between the switch and the receiving host are R_1 and R_2 , respectively. Assuming that the switch uses store-and-forward packet switching, what is the total end-to-end delay to send a packet of length L ? (Ignore queuing, propagation delay, and processing delay.)
- R12. What advantage does a circuit-switched network have over a packet-switched network? What advantages does TDM have over FDM in a circuit-switched network?
- R13. Suppose users share a 2 Mbps link. Also suppose each user transmits continuously at 1 Mbps when transmitting, but each user transmits only 20 percent of the time. (See the discussion of statistical multiplexing in Section 1.3.)

- a. When circuit switching is used, how many users can be supported?
- b. For the remainder of this problem, suppose packet switching is used. Why will there be essentially no queuing delay before the link if two or fewer users transmit at the same time? Why will there be a queuing delay if three users transmit at the same time?
- c. Find the probability that a given user is transmitting.
- d. Suppose now there are three users. Find the probability that at any given time, all three users are transmitting simultaneously. Find the fraction of time during which the queue grows.

R14. Why will two ISPs at the same level of the hierarchy often peer with each other? How does an IXP earn money?

R15. Some content providers have created their own networks. Describe Google's network. What motivates content providers to create these networks?

SECTION 1.4

- R16. Consider sending a packet from a source host to a destination host over a fixed route. List the delay components in the end-to-end delay. Which of these delays are constant and which are variable?
- R17. Visit the Transmission Versus Propagation Delay applet at the companion Web site. Among the rates, propagation delay, and packet sizes available, find a combination for which the sender finishes transmitting before the first bit of the packet reaches the receiver. Find another combination for which the first bit of the packet reaches the receiver before the sender finishes transmitting.
- R18. How long does it take a packet of length 1,000 bytes to propagate over a link of distance 2,500 km, propagation speed $2.5 \cdot 10^8$ m/s, and transmission rate 2 Mbps? More generally, how long does it take a packet of length L to propagate over a link of distance d , propagation speed s , and transmission rate R bps? Does this delay depend on packet length? Does this delay depend on transmission rate?
- R19. Suppose Host A wants to send a large file to Host B. The path from Host A to Host B has three links, of rates $R_1 = 500$ kbps, $R_2 = 2$ Mbps, and $R_3 = 1$ Mbps.
 - a. Assuming no other traffic in the network, what is the throughput for the file transfer?
 - b. Suppose the file is 4 million bytes. Dividing the file size by the throughput, roughly how long will it take to transfer the file to Host B?
 - c. Repeat (a) and (b), but now with R_2 reduced to 100 kbps.
- R20. Suppose end system A wants to send a large file to end system B. At a very high level, describe how end system A creates packets from the file. When

one of these packets arrives to a packet switch, what information in the packet does the switch use to determine the link onto which the packet is forwarded? Why is packet switching in the Internet analogous to driving from one city to another and asking directions along the way?

- R21. Visit the Queuing and Loss applet at the companion Web site. What is the maximum emission rate and the minimum transmission rate? With those rates, what is the traffic intensity? Run the applet with these rates and determine how long it takes for packet loss to occur. Then repeat the experiment a second time and determine again how long it takes for packet loss to occur. Are the values different? Why or why not?

SECTION 1.5

- R22. List five tasks that a layer can perform. Is it possible that one (or more) of these tasks could be performed by two (or more) layers?
- R23. What are the five layers in the Internet protocol stack? What are the principal responsibilities of each of these layers?
- R24. What is an application-layer message? A transport-layer segment? A network-layer datagram? A link-layer frame?
- R25. Which layers in the Internet protocol stack does a router process? Which layers does a link-layer switch process? Which layers does a host process?

SECTION 1.6

- R26. What is the difference between a virus and a worm?
- R27. Describe how a botnet can be created, and how it can be used for a DDoS attack.
- R28. Suppose Alice and Bob are sending packets to each other over a computer network. Suppose Trudy positions herself in the network so that she can capture all the packets sent by Alice and send whatever she wants to Bob; she can also capture all the packets sent by Bob and send whatever she wants to Alice. List some of the malicious things Trudy can do from this position.



Problems

- P1. Design and describe an application-level protocol to be used between an automatic teller machine and a bank's centralized computer. Your protocol should allow a user's card and password to be verified, the account balance (which is maintained at the centralized computer) to be queried, and an account withdrawal to be made (that is, money disbursed to the user). Your

protocol entities should be able to handle the all-too-common case in which there is not enough money in the account to cover the withdrawal. Specify your protocol by listing the messages exchanged and the action taken by the automatic teller machine or the bank's centralized computer on transmission and receipt of messages. Sketch the operation of your protocol for the case of a simple withdrawal with no errors, using a diagram similar to that in Figure 1.2. Explicitly state the assumptions made by your protocol about the underlying end-to-end transport service.

- P2. Equation 1.1 gives a formula for the end-to-end delay of sending one packet of length L over N links of transmission rate R . Generalize this formula for sending P such packets back-to-back over the N links.
- P3. Consider an application that transmits data at a steady rate (for example, the sender generates an N -bit unit of data every k time units, where k is small and fixed). Also, when such an application starts, it will continue running for a relatively long period of time. Answer the following questions, briefly justifying your answer:
- Would a packet-switched network or a circuit-switched network be more appropriate for this application? Why?
 - Suppose that a packet-switched network is used and the only traffic in this network comes from such applications as described above. Furthermore, assume that the sum of the application data rates is less than the capacities of each and every link. Is some form of congestion control needed? Why?
- P4. Consider the circuit-switched network in Figure 1.13. Recall that there are 4 circuits on each link. Label the four switches A, B, C and D, going in the clockwise direction.
- What is the maximum number of simultaneous connections that can be in progress at any one time in this network?
 - Suppose that all connections are between switches A and C. What is the maximum number of simultaneous connections that can be in progress?
 - Suppose we want to make four connections between switches A and C, and another four connections between switches B and D. Can we route these calls through the four links to accommodate all eight connections?
- P5. Review the car-caravan analogy in Section 1.4. Assume a propagation speed of 100 km/hour.
- Suppose the caravan travels 150 km, beginning in front of one tollbooth, passing through a second tollbooth, and finishing just after a third tollbooth. What is the end-to-end delay?
 - Repeat (a), now assuming that there are eight cars in the caravan instead of ten.



- P6. This elementary problem begins to explore propagation delay and transmission delay, two central concepts in data networking. Consider two hosts, A and B, connected by a single link of rate R bps. Suppose that the two hosts are separated by m meters, and suppose the propagation speed along the link is s meters/sec. Host A is to send a packet of size L bits to Host B.
- Express the propagation delay, d_{prop} , in terms of m and s .
 - Determine the transmission time of the packet, d_{trans} , in terms of L and R .
 - Ignoring processing and queuing delays, obtain an expression for the end-to-end delay.
 - Suppose Host A begins to transmit the packet at time $t = 0$. At time $t = d_{\text{trans}}$, where is the last bit of the packet?
 - Suppose d_{prop} is greater than d_{trans} . At time $t = d_{\text{trans}}$, where is the first bit of the packet?
 - Suppose d_{prop} is less than d_{trans} . At time $t = d_{\text{trans}}$, where is the first bit of the packet?
 - Suppose $s = 2.5 \cdot 10^8$, $L = 120$ bits, and $R = 56$ kbps. Find the distance m so that d_{prop} equals d_{trans} .
- P7. In this problem, we consider sending real-time voice from Host A to Host B over a packet-switched network (VoIP). Host A converts analog voice to a digital 64 kbps bit stream on the fly. Host A then groups the bits into 56-byte packets. There is one link between Hosts A and B; its transmission rate is 2 Mbps and its propagation delay is 10 msec. As soon as Host A gathers a packet, it sends it to Host B. As soon as Host B receives an entire packet, it converts the packet's bits to an analog signal. How much time elapses from the time a bit is created (from the original analog signal at Host A) until the bit is decoded (as part of the analog signal at Host B)?
- P8. Suppose users share a 3 Mbps link. Also suppose each user requires 150 kbps when transmitting, but each user transmits only 10 percent of the time. (See the discussion of packet switching versus circuit switching in Section 1.3.)
- When circuit switching is used, how many users can be supported?
 - For the remainder of this problem, suppose packet switching is used. Find the probability that a given user is transmitting.
 - Suppose there are 120 users. Find the probability that at any given time, exactly n users are transmitting simultaneously. (*Hint:* Use the binomial distribution.)
 - Find the probability that there are 21 or more users transmitting simultaneously.

- P9. Consider the discussion in Section 1.3 of packet switching versus circuit switching in which an example is provided with a 1 Mbps link. Users are generating data at a rate of 100 kbps when busy, but are busy generating data only with probability $p = 0.1$. Suppose that the 1 Mbps link is replaced by a 1 Gbps link.
- What is N , the maximum number of users that can be supported simultaneously under circuit switching?
 - Now consider packet switching and a user population of M users. Give a formula (in terms of p, M, N) for the probability that more than N users are sending data.
- P10. Consider a packet of length L which begins at end system A and travels over three links to a destination end system. These three links are connected by two packet switches. Let d_i , s_i , and R_i denote the length, propagation speed, and the transmission rate of link i , for $i = 1, 2, 3$. The packet switch delays each packet by d_{proc} . Assuming no queuing delays, in terms of d_i, s_i, R_i ($i = 1, 2, 3$), and L , what is the total end-to-end delay for the packet? Suppose now the packet is 1,500 bytes, the propagation speed on all three links is $2.5 \cdot 10^8$ m/s, the transmission rates of all three links are 2 Mbps, the packet switch processing delay is 3 msec, the length of the first link is 5,000 km, the length of the second link is 4,000 km, and the length of the last link is 1,000 km. For these values, what is the end-to-end delay?
- P11. In the above problem, suppose $R_1 = R_2 = R_3 = R$ and $d_{\text{proc}} = 0$. Further suppose the packet switch does not store-and-forward packets but instead immediately transmits each bit it receives before waiting for the entire packet to arrive. What is the end-to-end delay?
- P12. A packet switch receives a packet and determines the outbound link to which the packet should be forwarded. When the packet arrives, one other packet is halfway done being transmitted on this outbound link and four other packets are waiting to be transmitted. Packets are transmitted in order of arrival. Suppose all packets are 1,500 bytes and the link rate is 2 Mbps. What is the queuing delay for the packet? More generally, what is the queuing delay when all packets have length L , the transmission rate is R , x bits of the currently-being-transmitted packet have been transmitted, and n packets are already in the queue?
- P13. (a) Suppose N packets arrive simultaneously to a link at which no packets are currently being transmitted or queued. Each packet is of length L and the link has transmission rate R . What is the average queuing delay for the N packets?
- (b) Now suppose that N such packets arrive to the link every LN/R seconds. What is the average queuing delay of a packet?

- P14. Consider the queuing delay in a router buffer. Let I denote traffic intensity; that is, $I = La/R$. Suppose that the queuing delay takes the form $IL/R(1 - I)$ for $I < 1$.
- Provide a formula for the total delay, that is, the queuing delay plus the transmission delay.
 - Plot the total delay as a function of L/R .
- P15. Let a denote the rate of packets arriving at a link in packets/sec, and let μ denote the link's transmission rate in packets/sec. Based on the formula for the total delay (i.e., the queuing delay plus the transmission delay) derived in the previous problem, derive a formula for the total delay in terms of a and μ .
- P16. Consider a router buffer preceding an outbound link. In this problem, you will use Little's formula, a famous formula from queuing theory. Let N denote the average number of packets in the buffer plus the packet being transmitted. Let a denote the rate of packets arriving at the link. Let d denote the average total delay (i.e., the queuing delay plus the transmission delay) experienced by a packet. Little's formula is $N = a \cdot d$. Suppose that on average, the buffer contains 10 packets, and the average packet queuing delay is 10 msec. The link's transmission rate is 100 packets/sec. Using Little's formula, what is the average packet arrival rate, assuming there is no packet loss?
- P17.
- Generalize Equation 1.2 in Section 1.4.3 for heterogeneous processing rates, transmission rates, and propagation delays.
 - Repeat (a), but now also suppose that there is an average queuing delay of d_{queue} at each node.
- P18.
- Perform a Traceroute between source and destination on the same continent at three different hours of the day.
 - Find the average and standard deviation of the round-trip delays at each of the three hours.
 - Find the number of routers in the path at each of the three hours. Did the paths change during any of the hours?
 - Try to identify the number of ISP networks that the Traceroute packets pass through from source to destination. Routers with similar names and/or similar IP addresses should be considered as part of the same ISP. In your experiments, do the largest delays occur at the peering interfaces between adjacent ISPs?
 - Repeat the above for a source and destination on different continents. Compare the intra-continent and inter-continent results.
- P19.
- Visit the site www.traceroute.org and perform traceroutes from two different cities in France to the same destination host in the United States. How many links are the same in the two traceroutes? Is the transatlantic link the same?



- (b) Repeat (a) but this time choose one city in France and another city in Germany.
- (c) Pick a city in the United States, and perform traceroutes to two hosts, each in a different city in China. How many links are common in the two traceroutes? Do the two traceroutes diverge before reaching China?
- P20. Consider the throughput example corresponding to Figure 1.20(b). Now suppose that there are M client-server pairs rather than 10. Denote R_s , R_c , and R for the rates of the server links, client links, and network link. Assume all other links have abundant capacity and that there is no other traffic in the network besides the traffic generated by the M client-server pairs. Derive a general expression for throughput in terms of R_s , R_c , R , and M .
- P21. Consider Figure 1.19(b). Now suppose that there are M paths between the server and the client. No two paths share any link. Path k ($k = 1, \dots, M$) consists of N links with transmission rates $R_1^k, R_2^k, \dots, R_N^k$. If the server can only use one path to send data to the client, what is the maximum throughput that the server can achieve? If the server can use all M paths to send data, what is the maximum throughput that the server can achieve?
- P22. Consider Figure 1.19(b). Suppose that each link between the server and the client has a packet loss probability p , and the packet loss probabilities for these links are independent. What is the probability that a packet (sent by the server) is successfully received by the receiver? If a packet is lost in the path from the server to the client, then the server will re-transmit the packet. On average, how many times will the server re-transmit the packet in order for the client to successfully receive the packet?
- P23. Consider Figure 1.19(a). Assume that we know the bottleneck link along the path from the server to the client is the first link with rate R_s bits/sec. Suppose we send a pair of packets back to back from the server to the client, and there is no other traffic on this path. Assume each packet of size L bits, and both links have the same propagation delay d_{prop} .
- What is the packet inter-arrival time at the destination? That is, how much time elapses from when the last bit of the first packet arrives until the last bit of the second packet arrives?
 - Now assume that the second link is the bottleneck link (i.e., $R_c < R_s$). Is it possible that the second packet queues at the input queue of the second link? Explain. Now suppose that the server sends the second packet T seconds after sending the first packet. How large must T be to ensure no queuing before the second link? Explain.
- P24. Suppose you would like to urgently deliver 40 terabytes data from Boston to Los Angeles. You have available a 100 Mbps dedicated link for data transfer. Would you prefer to transmit the data via this link or instead use FedEx overnight delivery? Explain.

- P25. Suppose two hosts, A and B, are separated by 20,000 kilometers and are connected by a direct link of $R = 2$ Mbps. Suppose the propagation speed over the link is $2.5 \cdot 10^8$ meters/sec.
- Calculate the bandwidth-delay product, $R \cdot d_{\text{prop}}$.
 - Consider sending a file of 800,000 bits from Host A to Host B. Suppose the file is sent continuously as one large message. What is the maximum number of bits that will be in the link at any given time?
 - Provide an interpretation of the bandwidth-delay product.
 - What is the width (in meters) of a bit in the link? Is it longer than a football field?
 - Derive a general expression for the width of a bit in terms of the propagation speed s , the transmission rate R , and the length of the link m .
- P26. Referring to problem P25, suppose we can modify R . For what value of R is the width of a bit as long as the length of the link?
- P27. Consider problem P25 but now with a link of $R = 1$ Gbps.
- Calculate the bandwidth-delay product, $R \cdot d_{\text{prop}}$.
 - Consider sending a file of 800,000 bits from Host A to Host B. Suppose the file is sent continuously as one big message. What is the maximum number of bits that will be in the link at any given time?
 - What is the width (in meters) of a bit in the link?
- P28. Refer again to problem P25.
- How long does it take to send the file, assuming it is sent continuously?
 - Suppose now the file is broken up into 20 packets with each packet containing 40,000 bits. Suppose that each packet is acknowledged by the receiver and the transmission time of an acknowledgment packet is negligible. Finally, assume that the sender cannot send a packet until the preceding one is acknowledged. How long does it take to send the file?
 - Compare the results from (a) and (b).
- P29. Suppose there is a 10 Mbps microwave link between a geostationary satellite and its base station on Earth. Every minute the satellite takes a digital photo and sends it to the base station. Assume a propagation speed of $2.4 \cdot 10^8$ meters/sec.
- What is the propagation delay of the link?
 - What is the bandwidth-delay product, $R \cdot d_{\text{prop}}$?
 - Let x denote the size of the photo. What is the minimum value of x for the microwave link to be continuously transmitting?
- P30. Consider the airline travel analogy in our discussion of layering in Section 1.5, and the addition of headers to protocol data units as they flow down

the protocol stack. Is there an equivalent notion of header information that is added to passengers and baggage as they move down the airline protocol stack?

- P31. In modern packet-switched networks, including the Internet, the source host segments long, application-layer messages (for example, an image or a music file) into smaller packets and sends the packets into the network. The receiver then reassembles the packets back into the original message. We refer to this process as *message segmentation*. Figure 1.27 illustrates the end-to-end transport of a message with and without message segmentation. Consider a message that is $8 \cdot 10^6$ bits long that is to be sent from source to destination in Figure 1.27. Suppose each link in the figure is 2 Mbps. Ignore propagation, queuing, and processing delays.

- Consider sending the message from source to destination *without* message segmentation. How long does it take to move the message from the source host to the first packet switch? Keeping in mind that each switch uses store-and-forward packet switching, what is the total time to move the message from source host to destination host?
- Now suppose that the message is segmented into 800 packets, with each packet being 10,000 bits long. How long does it take to move the first packet from source host to the first switch? When the first packet is being sent from the first switch to the second switch, the second packet is being sent from the source host to the first switch. At what time will the second packet be fully received at the first switch?
- How long does it take to move the file from source host to destination host when message segmentation is used? Compare this result with your answer in part (a) and comment.

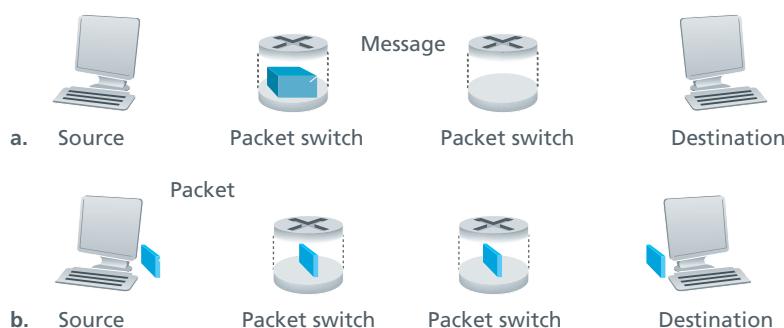


Figure 1.27 End-to-end message transport: (a) without message segmentation; (b) with message segmentation

- d. In addition to reducing delay, what are reasons to use message segmentation?
 - e. Discuss the drawbacks of message segmentation.
- P32. Experiment with the Message Segmentation applet at the book's Web site. Do the delays in the applet correspond to the delays in the previous problem? How do link propagation delays affect the overall end-to-end delay for packet switching (with message segmentation) and for message switching?
- P33. Consider sending a large file of F bits from Host A to Host B. There are three links (and two switches) between A and B, and the links are uncongested (that is, no queuing delays). Host A segments the file into segments of S bits each and adds 80 bits of header to each segment, forming packets of $L = 80 + S$ bits. Each link has a transmission rate of R bps. Find the value of S that minimizes the delay of moving the file from Host A to Host B. Disregard propagation delay.
- P34. Skype offers a service that allows you to make a phone call from a PC to an ordinary phone. This means that the voice call must pass through both the Internet and through a telephone network. Discuss how this might be done.



Wireshark Lab

“Tell me and I forget. Show me and I remember. Involve me and I understand.”
Chinese proverb

One's understanding of network protocols can often be greatly deepened by seeing them in action and by playing around with them—observing the sequence of messages exchanged between two protocol entities, delving into the details of protocol operation, causing protocols to perform certain actions, and observing these actions and their consequences. This can be done in simulated scenarios or in a real network environment such as the Internet. The Java applets at the textbook Web site take the first approach. In the Wireshark labs, we'll take the latter approach. You'll run network applications in various scenarios using a computer on your desk, at home, or in a lab. You'll observe the network protocols in your computer, interacting and exchanging messages with protocol entities executing elsewhere in the Internet. Thus, you and your computer will be an integral part of these live labs. You'll observe—and you'll learn—by doing.

The basic tool for observing the messages exchanged between executing protocol entities is called a **packet sniffer**. As the name suggests, a packet sniffer passively copies (sniffs) messages being sent from and received by your computer; it also displays the contents of the various protocol fields of these captured messages. A screenshot of the Wireshark packet sniffer is shown in Figure 1.28. Wireshark is a free packet sniffer that runs on Windows, Linux/Unix, and Mac

Command menus

Listing of captured packets

Details of selected packet header

Packet contents in hexadecimal and ASCII

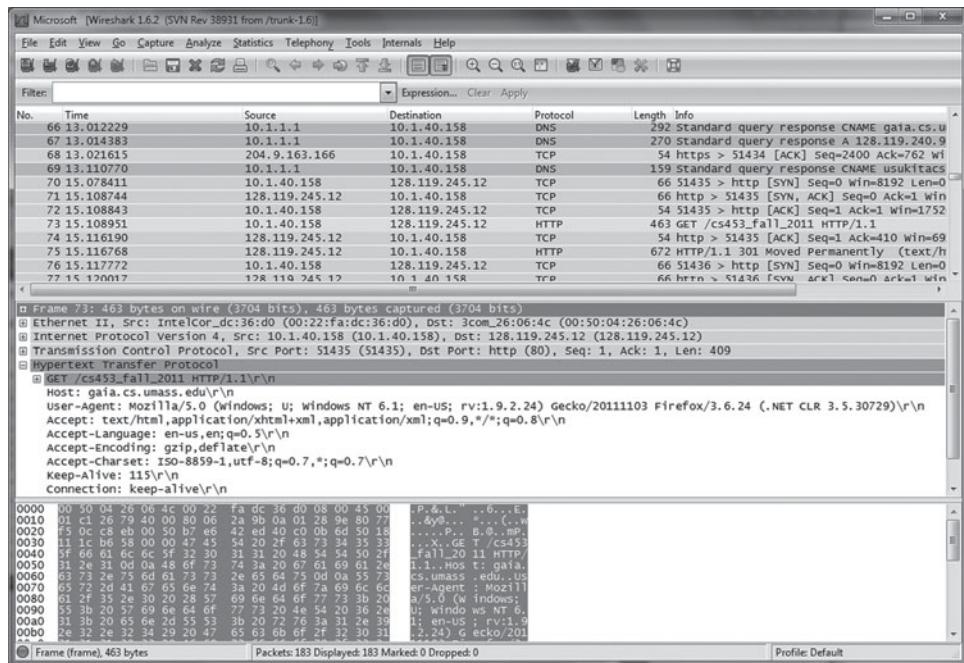


Figure 1.28 ♦ A Wireshark screen shot (Wireshark screenshot reprinted by permission of the Wireshark Foundation.)

computers. Throughout the textbook, you will find Wireshark labs that allow you to explore a number of the protocols studied in the chapter. In this first Wireshark lab, you'll obtain and install a copy of Wireshark, access a Web site, and capture and examine the protocol messages being exchanged between your Web browser and the Web server.

You can find full details about this first Wireshark lab (including instructions about how to obtain and install Wireshark) at the Web site <http://www.awl.com/kurose-ross>.

AN INTERVIEW WITH...

Leonard Kleinrock

Leonard Kleinrock is a professor of computer science at the University of California, Los Angeles. In 1969, his computer at UCLA became the first node of the Internet. His creation of packet-switching principles in 1961 became the technology behind the Internet. He received his B.E.E. from the City College of New York (CCNY) and his masters and PhD in electrical engineering from MIT.



What made you decide to specialize in networking/Internet technology?

As a PhD student at MIT in 1959, I looked around and found that most of my classmates were doing research in the area of information theory and coding theory. At MIT, there was the great researcher, Claude Shannon, who had launched these fields and had solved most of the important problems already. The research problems that were left were hard and of lesser consequence. So I decided to launch out in a new area that no one else had yet conceived of. Remember that at MIT I was surrounded by lots of computers, and it was clear to me that soon these machines would need to communicate with each other. At the time, there was no effective way for them to do so, so I decided to develop the technology that would permit efficient and reliable data networks to be created.

What was your first job in the computer industry? What did it entail?

I went to the evening session at CCNY from 1951 to 1957 for my bachelor's degree in electrical engineering. During the day, I worked first as a technician and then as an engineer at a small, industrial electronics firm called Photobell. While there, I introduced digital technology to their product line. Essentially, we were using photoelectric devices to detect the presence of certain items (boxes, people, etc.) and the use of a circuit known then as a *bistable multivibrator* was just the kind of technology we needed to bring digital processing into this field of detection. These circuits happen to be the building blocks for computers, and have come to be known as *flip-flops* or *switches* in today's vernacular.

What was going through your mind when you sent the first host-to-host message (from UCLA to the Stanford Research Institute)?

Frankly, we had no idea of the importance of that event. We had not prepared a special message of historic significance, as did so many inventors of the past (Samuel Morse with "What hath God wrought." or Alexander Graham Bell with "Watson, come here! I want you." or Neal Armstrong with "That's one small step for a man, one giant leap for mankind.") Those guys were *smart!* They understood media and public relations. All we wanted to do was to login to the SRI computer. So we typed the "L", which was correctly received, we typed the "o" which was received, and then we typed the "g" which caused the SRI host computer to

crash! So, it turned out that our message was the shortest and perhaps the most prophetic message ever, namely “Lo!” as in “Lo and behold!”

Earlier that year, I was quoted in a UCLA press release saying that once the network was up and running, it would be possible to gain access to computer utilities from our homes and offices as easily as we gain access to electricity and telephone connectivity. So my vision at that time was that the Internet would be ubiquitous, always on, always available, anyone with any device could connect from any location, and it would be invisible. However, I never anticipated that my 99-year-old mother would use the Internet—and indeed she did!

What is your vision for the future of networking?

The easy part of the vision is to predict the infrastructure itself. I anticipate that we see considerable deployment of nomadic computing, mobile devices, and smart spaces. Indeed, the availability of lightweight, inexpensive, high-performance, portable computing, and communication devices (plus the ubiquity of the Internet) has enabled us to become nomads.

Nomadic computing refers to the technology that enables end users who travel from place to place to gain access to Internet services in a transparent fashion, no matter where they travel and no matter what device they carry or gain access to. The harder part of the vision is to predict the applications and services, which have consistently surprised us in dramatic ways (email, search technologies, the world-wide-web, blogs, social networks, user generation, and sharing of music, photos, and videos, etc.). We are on the verge of a new class of surprising and innovative mobile applications delivered to our hand-held devices.

The next step will enable us to move out from the netherworld of cyberspace to the physical world of smart spaces. Our environments (desks, walls, vehicles, watches, belts, and so on) will come alive with technology, through actuators, sensors, logic, processing, storage, cameras, microphones, speakers, displays, and communication. This embedded technology will allow our environment to provide the IP services we want. When I walk into a room, the room will know I entered. I will be able to communicate with my environment naturally, as in spoken English; my requests will generate replies that present Web pages to me from wall displays, through my eyeglasses, as speech, holograms, and so forth.

Looking a bit further out, I see a networking future that includes the following additional key components. I see intelligent software agents deployed across the network whose function it is to mine data, act on that data, observe trends, and carry out tasks dynamically and adaptively. I see considerably more network traffic generated not so much by humans, but by these embedded devices and these intelligent software agents. I see large collections of self-organizing systems controlling this vast, fast network. I see huge amounts of information flashing across this network instantaneously with this information undergoing enormous processing and filtering. The Internet will essentially be a pervasive global nervous system. I see all these things and more as we move headlong through the twenty-first century.

What people have inspired you professionally?

By far, it was Claude Shannon from MIT, a brilliant researcher who had the ability to relate his mathematical ideas to the physical world in highly intuitive ways. He was on my PhD thesis committee.

Do you have any advice for students entering the networking/Internet field?

The Internet and all that it enables is a vast new frontier, full of amazing challenges. There is room for great innovation. Don't be constrained by today's technology. Reach out and imagine what could be and then make it happen.

Application Layer

Network applications are the *raisons d'être* of a computer network—if we couldn't conceive of any useful applications, there wouldn't be any need for networking protocols that support these applications. Since the Internet's inception, numerous useful and entertaining applications have indeed been created. These applications have been the driving force behind the Internet's success, motivating people in homes, schools, governments, and businesses to make the Internet an integral part of their daily activities.

Internet applications include the classic text-based applications that became popular in the 1970s and 1980s: text email, remote access to computers, file transfers, and newsgroups. They include the killer application of the mid-1990s, the World Wide Web, encompassing Web surfing, search, and electronic commerce. They include instant messaging and P2P file sharing, the two killer applications introduced at the end of the millennium. Since 2000, we have seen an explosion of popular voice and video applications, including: voice-over-IP (VoIP) and video conferencing over IP such as Skype; user-generated video distribution such as YouTube; and movies on demand such as Netflix. During this same period we have also seen the immergence of highly engaging multi-player online games, including Second Life and World of Warcraft. And most recently, we have seen the emergence of a new generation of social networking applications, such as Facebook and Twitter, which have created engaging human networks on top of the Internet's network of routers and communication links. Clearly, there has been no slowing down of new

and exciting Internet applications. Perhaps some of the readers of this text will create the next generation of killer Internet applications!

In this chapter we study the conceptual and implementation aspects of network applications. We begin by defining key application-layer concepts, including network services required by applications, clients and servers, processes, and transport-layer interfaces. We examine several network applications in detail, including the Web, e-mail, DNS, and peer-to-peer (P2P) file distribution (Chapter 8 focuses on multimedia applications, including streaming video and VoIP). We then cover network application development, over both TCP and UDP. In particular, we study the socket API and walk through some simple client-server applications in Python. We also provide several fun and interesting socket programming assignments at the end of the chapter.

The application layer is a particularly good place to start our study of protocols. It's familiar ground. We're acquainted with many of the applications that rely on the protocols we'll study. It will give us a good feel for what protocols are all about and will introduce us to many of the same issues that we'll see again when we study transport, network, and link layer protocols.

2.1 Principles of Network Applications

Suppose you have an idea for a new network application. Perhaps this application will be a great service to humanity, or will please your professor, or will bring you great wealth, or will simply be fun to develop. Whatever the motivation may be, let's now examine how you transform the idea into a real-world network application.

At the core of network application development is writing programs that run on different end systems and communicate with each other over the network. For example, in the Web application there are two distinct programs that communicate with each other: the browser program running in the user's host (desktop, laptop, tablet, smartphone, and so on); and the Web server program running in the Web server host. As another example, in a P2P file-sharing system there is a program in each host that participates in the file-sharing community. In this case, the programs in the various hosts may be similar or identical.

Thus, when developing your new application, you need to write software that will run on multiple end systems. This software could be written, for example, in C, Java, or Python. Importantly, you do not need to write software that runs on network-core devices, such as routers or link-layer switches. Even if you wanted to write application software for these network-core devices, you wouldn't be able to do so. As we learned in Chapter 1, and as shown earlier in Figure 1.24, network-core devices do not function at the application layer but instead function at lower layers—specifically at the network layer and below. This basic design—namely, confining application software to the end systems—as shown in Figure 2.1, has facilitated the rapid development and deployment of a vast array of network applications.

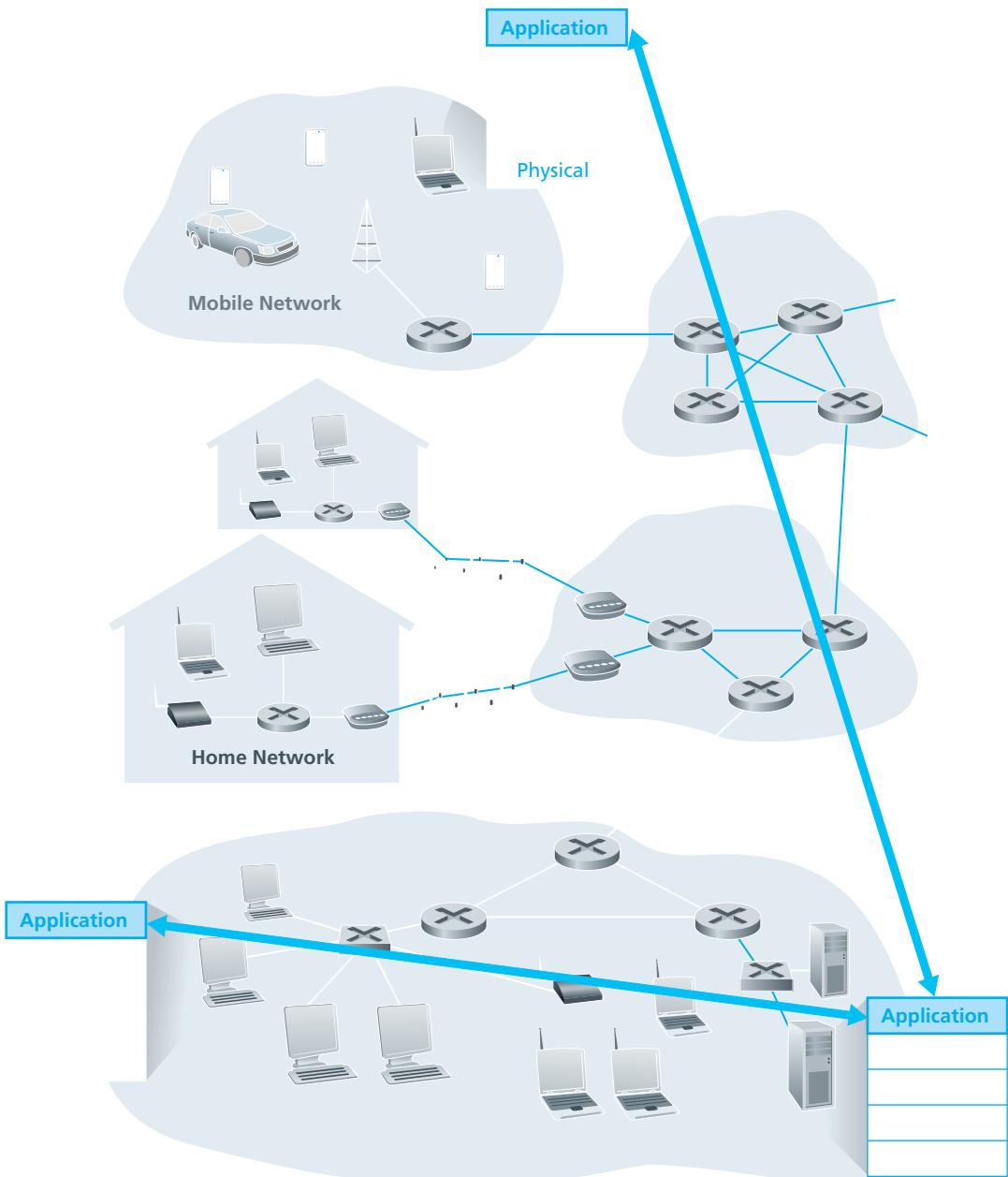


Figure 2.1 ♦ Communication for a network application takes place between end systems at the application layer

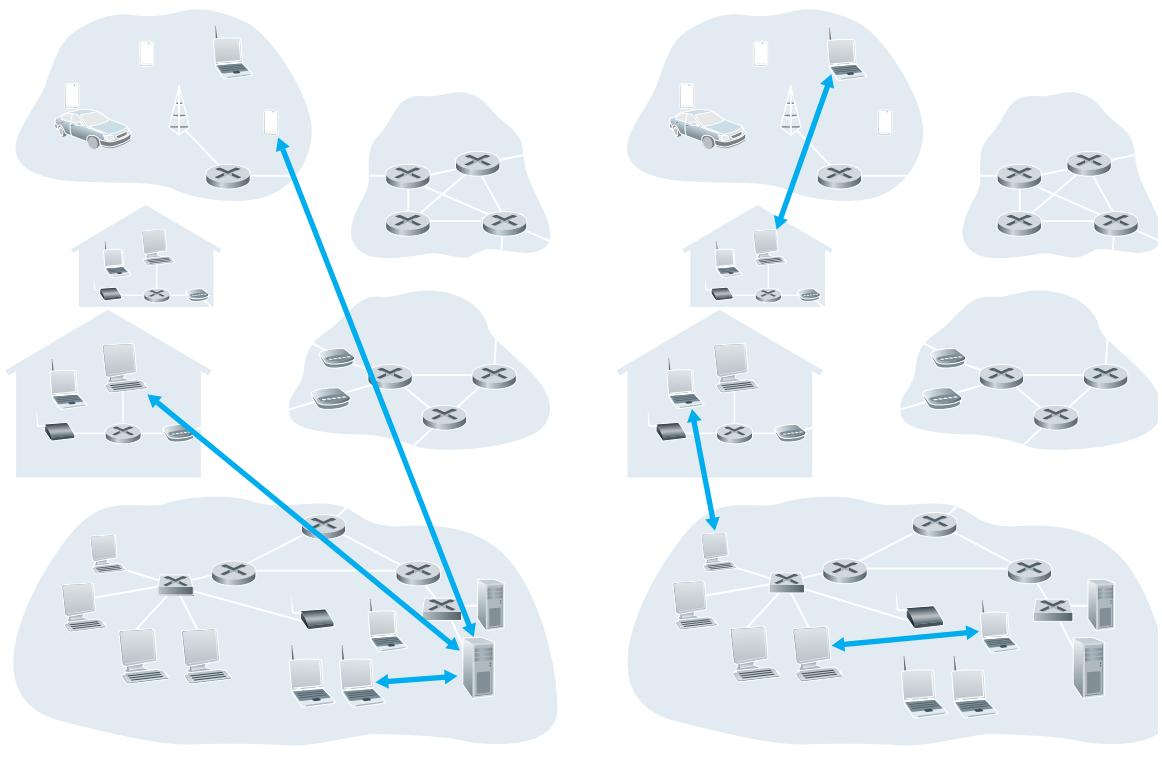
2.1.1 Network Application Architectures

Before diving into software coding, you should have a broad architectural plan for your application. Keep in mind that an application's architecture is distinctly different from the network architecture (e.g., the five-layer Internet architecture discussed in Chapter 1). From the application developer's perspective, the network architecture is fixed and provides a specific set of services to applications. The **application architecture**, on the other hand, is designed by the application developer and dictates how the application is structured over the various end systems. In choosing the application architecture, an application developer will likely draw on one of the two predominant architectural paradigms used in modern network applications: the client-server architecture or the peer-to-peer (P2P) architecture.

In a **client-server architecture**, there is an always-on host, called the *server*, which services requests from many other hosts, called *clients*. A classic example is the Web application for which an always-on Web server services requests from browsers running on client hosts. When a Web server receives a request for an object from a client host, it responds by sending the requested object to the client host. Note that with the client-server architecture, clients do not directly communicate with each other; for example, in the Web application, two browsers do not directly communicate. Another characteristic of the client-server architecture is that the server has a fixed, well-known address, called an IP address (which we'll discuss soon). Because the server has a fixed, well-known address, and because the server is always on, a client can always contact the server by sending a packet to the server's IP address. Some of the better-known applications with a client-server architecture include the Web, FTP, Telnet, and e-mail. The client-server architecture is shown in Figure 2.2(a).

Often in a client-server application, a single-server host is incapable of keeping up with all the requests from clients. For example, a popular social-networking site can quickly become overwhelmed if it has only one server handling all of its requests. For this reason, a **data center**, housing a large number of hosts, is often used to create a powerful virtual server. The most popular Internet services—such as search engines (e.g., Google and Bing), Internet commerce (e.g., Amazon and e-Bay), Web-based email (e.g., Gmail and Yahoo Mail), social networking (e.g., Facebook and Twitter)—employ one or more data centers. As discussed in Section 1.3.3, Google has 30 to 50 data centers distributed around the world, which collectively handle search, YouTube, Gmail, and other services. A data center can have hundreds of thousands of servers, which must be powered and maintained. Additionally, the service providers must pay recurring interconnection and bandwidth costs for sending data from their data centers.

In a **P2P architecture**, there is minimal (or no) reliance on dedicated servers in data centers. Instead the application exploits direct communication between pairs of intermittently connected hosts, called *peers*. The peers are not owned by the service provider, but are instead desktops and laptops controlled by users, with most of the peers residing in homes, universities, and offices. Because the peers communicate without passing through a dedicated server, the architecture is called peer-to-peer. Many of today's most popular and traffic-intensive applications are based on P2P architectures. These applications include file sharing (e.g., BitTorrent), peer-assisted



a. Client-server architecture

b. Peer-to-peer architecture

Figure 2.2 ♦ (a) Client-server architecture; (b) P2P architecture

download acceleration (e.g., Xunlei), Internet Telephony (e.g., Skype), and IPTV (e.g., Kankan and PPstream). The P2P architecture is illustrated in Figure 2.2(b). We mention that some applications have hybrid architectures, combining both client-server and P2P elements. For example, for many instant messaging applications, servers are used to track the IP addresses of users, but user-to-user messages are sent directly between user hosts (without passing through intermediate servers).

One of the most compelling features of P2P architectures is their **self-scalability**. For example, in a P2P file-sharing application, although each peer generates workload by requesting files, each peer also adds service capacity to the system by distributing files to other peers. P2P architectures are also cost effective, since they normally don't require significant server infrastructure and server bandwidth (in contrast with clients-server designs with datacenters). However, future P2P applications face three major challenges:

1. **ISP Friendly.** Most residential ISPs (including DSL and cable ISPs) have been dimensioned for “asymmetrical” bandwidth usage, that is, for much more

downstream than upstream traffic. But P2P video streaming and file distribution applications shift upstream traffic from servers to residential ISPs, thereby putting significant stress on the ISPs. Future P2P applications need to be designed so that they are friendly to ISPs [Xie 2008].

2. *Security.* Because of their highly distributed and open nature, P2P applications can be a challenge to secure [Doucer 2002; Yu 2006; Liang 2006; Naoumov 2006; Dhungel 2008; LeBlond 2011].
3. *Incentives.* The success of future P2P applications also depends on convincing users to volunteer bandwidth, storage, and computation resources to the applications, which is the challenge of incentive design [Feldman 2005; Piatek 2008; Aperjis 2008; Liu 2010].

2.1.2 Processes Communicating

Before building your network application, you also need a basic understanding of how the programs, running in multiple end systems, communicate with each other. In the jargon of operating systems, it is not actually programs but **processes** that communicate. A process can be thought of as a program that is running within an end system. When processes are running on the same end system, they can communicate with each other with interprocess communication, using rules that are governed by the end system's operating system. But in this book we are not particularly interested in how processes in the same host communicate, but instead in how processes running on *different* hosts (with potentially different operating systems) communicate.

Processes on two different end systems communicate with each other by exchanging **messages** across the computer network. A sending process creates and sends messages into the network; a receiving process receives these messages and possibly responds by sending messages back. Figure 2.1 illustrates that processes communicating with each other reside in the application layer of the five-layer protocol stack.

Client and Server Processes

A network application consists of pairs of processes that send messages to each other over a network. For example, in the Web application a client browser process exchanges messages with a Web server process. In a P2P file-sharing system, a file is transferred from a process in one peer to a process in another peer. For each pair of communicating processes, we typically label one of the two processes as the **client** and the other process as the **server**. With the Web, a browser is a client process and a Web server is a server process. With P2P file sharing, the peer that is downloading the file is labeled as the client, and the peer that is uploading the file is labeled as the server.

You may have observed that in some applications, such as in P2P file sharing, a process can be both a client and a server. Indeed, a process in a P2P file-sharing system can both upload and download files. Nevertheless, in the context of any given

communication session between a pair of processes, we can still label one process as the client and the other process as the server. We define the client and server processes as follows:

*In the context of a communication session between a pair of processes, the process that initiates the communication (that is, initially contacts the other process at the beginning of the session) is labeled as the **client**. The process that waits to be contacted to begin the session is the **server**.*

In the Web, a browser process initializes contact with a Web server process; hence the browser process is the client and the Web server process is the server. In P2P file sharing, when Peer A asks Peer B to send a specific file, Peer A is the client and Peer B is the server in the context of this specific communication session. When there's no confusion, we'll sometimes also use the terminology "client side and server side of an application." At the end of this chapter, we'll step through simple code for both the client and server sides of network applications.

The Interface Between the Process and the Computer Network

As noted above, most applications consist of pairs of communicating processes, with the two processes in each pair sending messages to each other. Any message sent from one process to another must go through the underlying network. A process sends messages into, and receives messages from, the network through a software interface called a **socket**. Let's consider an analogy to help us understand processes and sockets. A process is analogous to a house and its socket is analogous to its door. When a process wants to send a message to another process on another host, it shoves the message out its door (socket). This sending process assumes that there is a transportation infrastructure on the other side of its door that will transport the message to the door of the destination process. Once the message arrives at the destination host, the message passes through the receiving process's door (socket), and the receiving process then acts on the message.

Figure 2.3 illustrates socket communication between two processes that communicate over the Internet. (Figure 2.3 assumes that the underlying transport protocol used by the processes is the Internet's TCP protocol.) As shown in this figure, a socket is the interface between the application layer and the transport layer within a host. It is also referred to as the **Application Programming Interface (API)** between the application and the network, since the socket is the programming interface with which network applications are built. The application developer has control of everything on the application-layer side of the socket but has little control of the transport-layer side of the socket. The only control that the application developer has on the transport-layer side is (1) the choice of transport protocol and (2) perhaps the ability to fix a few transport-layer parameters such as maximum buffer and maximum segment sizes (to be covered in Chapter 3). Once the application developer chooses a transport protocol (if a choice is available),

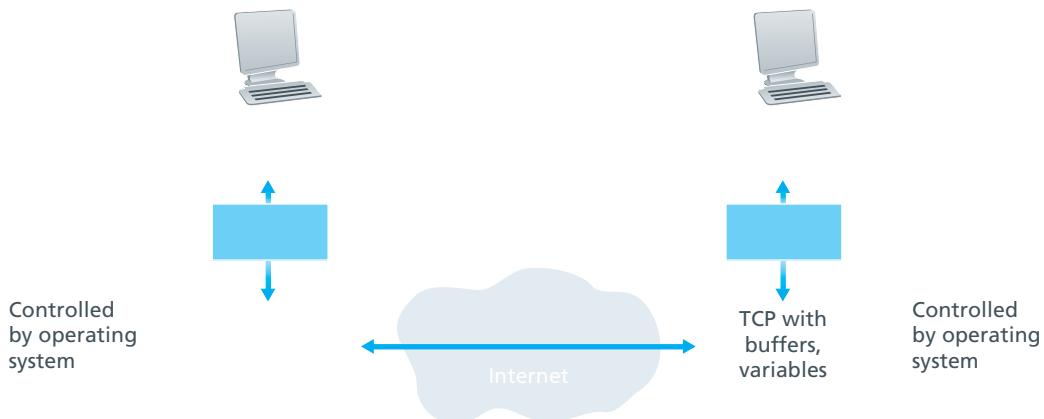


Figure 2.3 ♦ Application processes, sockets, and underlying transport protocol

the application is built using the transport-layer services provided by that protocol. We'll explore sockets in some detail in Section 2.7.

Addressing Processes

In order to send postal mail to a particular destination, the destination needs to have an address. Similarly, in order for a process running on one host to send packets to a process running on another host, the receiving process needs to have an address. To identify the receiving process, two pieces of information need to be specified: (1) the address of the host and (2) an identifier that specifies the receiving process in the destination host.

In the Internet, the host is identified by its **IP address**. We'll discuss IP addresses in great detail in Chapter 4. For now, all we need to know is that an IP address is a 32-bit quantity that we can think of as uniquely identifying the host. In addition to knowing the address of the host to which a message is destined, the sending process must also identify the receiving process (more specifically, the receiving socket) running in the host. This information is needed because in general a host could be running many network applications. A destination **port number** serves this purpose. Popular applications have been assigned specific port numbers. For example, a Web server is identified by port number 80. A mail server process (using the SMTP protocol) is identified by port number 25. A list of well-known port numbers for all Internet standard protocols can be found at <http://www.iana.org>. We'll examine port numbers in detail in Chapter 3.

2.1.3 Transport Services Available to Applications

Recall that a socket is the interface between the application process and the transport-layer protocol. The application at the sending side pushes messages through the socket. At the other side of the socket, the transport-layer protocol has the responsibility of getting the messages to the socket of the receiving process.

Many networks, including the Internet, provide more than one transport-layer protocol. When you develop an application, you must choose one of the available transport-layer protocols. How do you make this choice? Most likely, you would study the services provided by the available transport-layer protocols, and then pick the protocol with the services that best match your application's needs. The situation is similar to choosing either train or airplane transport for travel between two cities. You have to choose one or the other, and each transportation mode offers different services. (For example, the train offers downtown pickup and drop-off, whereas the plane offers shorter travel time.)

What are the services that a transport-layer protocol can offer to applications invoking it? We can broadly classify the possible services along four dimensions: reliable data transfer, throughput, timing, and security.

Reliable Data Transfer

As discussed in Chapter 1, packets can get lost within a computer network. For example, a packet can overflow a buffer in a router, or can be discarded by a host or router after having some of its bits corrupted. For many applications—such as electronic mail, file transfer, remote host access, Web document transfers, and financial applications—data loss can have devastating consequences (in the latter case, for either the bank or the customer!). Thus, to support these applications, something has to be done to guarantee that the data sent by one end of the application is delivered correctly and completely to the other end of the application. If a protocol provides such a guaranteed data delivery service, it is said to provide **reliable data transfer**. One important service that a transport-layer protocol can potentially provide to an application is process-to-process reliable data transfer. When a transport protocol provides this service, the sending process can just pass its data into the socket and know with complete confidence that the data will arrive without errors at the receiving process.

When a transport-layer protocol doesn't provide reliable data transfer, some of the data sent by the sending process may never arrive at the receiving process. This may be acceptable for **loss-tolerant applications**, most notably multimedia applications such as conversational audio/video that can tolerate some amount of data loss. In these multimedia applications, lost data might result in a small glitch in the audio/video—not a crucial impairment.

Throughput

In Chapter 1 we introduced the concept of available throughput, which, in the context of a communication session between two processes along a network path, is the rate at which the sending process can deliver bits to the receiving process. Because other sessions will be sharing the bandwidth along the network path, and because these other sessions will be coming and going, the available throughput can fluctuate with time. These observations lead to another natural service that a transport-layer protocol could provide, namely, guaranteed available throughput at some specified rate. With such a service, the application could request a guaranteed throughput of r bits/sec, and the transport protocol would then ensure that the available throughput is always at least r bits/sec. Such a guaranteed throughput service would appeal to many applications. For example, if an Internet telephony application encodes voice at 32 kbps, it needs to send data into the network and have data delivered to the receiving application at this rate. If the transport protocol cannot provide this throughput, the application would need to encode at a lower rate (and receive enough throughput to sustain this lower coding rate) or may have to give up, since receiving, say, half of the needed throughput is of little or no use to this Internet telephony application. Applications that have throughput requirements are said to be **bandwidth-sensitive applications**. Many current multimedia applications are bandwidth sensitive, although some multimedia applications may use adaptive coding techniques to encode digitized voice or video at a rate that matches the currently available throughput.

While bandwidth-sensitive applications have specific throughput requirements, **elastic applications** can make use of as much, or as little, throughput as happens to be available. Electronic mail, file transfer, and Web transfers are all elastic applications. Of course, the more throughput, the better. There's an adage that says that one cannot be too rich, too thin, or have too much throughput!

Timing

A transport-layer protocol can also provide timing guarantees. As with throughput guarantees, timing guarantees can come in many shapes and forms. An example guarantee might be that every bit that the sender pumps into the socket arrives at the receiver's socket no more than 100 msec later. Such a service would be appealing to interactive real-time applications, such as Internet telephony, virtual environments, teleconferencing, and multiplayer games, all of which require tight timing constraints on data delivery in order to be effective. (See Chapter 7, [Gauthier 1999; Ramjee 1994].) Long delays in Internet telephony, for example, tend to result in unnatural pauses in the conversation; in a multiplayer game or virtual interactive environment, a long delay between taking an action and seeing the response from the environment (for example, from another player at the end of an end-to-end connection) makes the application feel less realistic. For non-real-time applications,

lower delay is always preferable to higher delay, but no tight constraint is placed on the end-to-end delays.

Security

Finally, a transport protocol can provide an application with one or more security services. For example, in the sending host, a transport protocol can encrypt all data transmitted by the sending process, and in the receiving host, the transport-layer protocol can decrypt the data before delivering the data to the receiving process. Such a service would provide confidentiality between the two processes, even if the data is somehow observed between sending and receiving processes. A transport protocol can also provide other security services in addition to confidentiality, including data integrity and end-point authentication, topics that we'll cover in detail in Chapter 8.

2.1.4 Transport Services Provided by the Internet

Up until this point, we have been considering transport services that a computer network *could* provide in general. Let's now get more specific and examine the type of transport services provided by the Internet. The Internet (and, more generally, TCP/IP networks) makes two transport protocols available to applications, UDP and TCP. When you (as an application developer) create a new network application for the Internet, one of the first decisions you have to make is whether to use UDP or TCP. Each of these protocols offers a different set of services to the invoking applications. Figure 2.4 shows the service requirements for some selected applications.

Application	Data Loss	Throughput	Time-Sensitive
File transfer/download	No loss	Elastic	No
E-mail	No loss	Elastic	No
Web documents	No loss	Elastic (few kbps)	No
Internet telephony/ Video conferencing	Loss-tolerant	Audio: few kbps–1 Mbps Video: 10 kbps–5 Mbps	Yes: 100s of msec
Streaming stored audio/video	Loss-tolerant	Same as above	Yes: few seconds
Interactive games	Loss-tolerant	Few kbps–10 kbps	Yes: 100s of msec
Instant messaging	No loss	Elastic	Yes and no

Figure 2.4 ♦ Requirements of selected network applications

TCP Services

The TCP service model includes a connection-oriented service and a reliable data transfer service. When an application invokes TCP as its transport protocol, the application receives both of these services from TCP.

- *Connection-oriented service.* TCP has the client and server exchange transport-layer control information with each other *before* the application-level messages begin to flow. This so-called handshaking procedure alerts the client and server, allowing them to prepare for an onslaught of packets. After the handshaking phase, a **TCP connection** is said to exist between the sockets of the two processes. The connection is a full-duplex connection in that the two processes can send messages to each other over the connection at the same time. When the application finishes sending messages, it must tear down the connection. In Chapter 3 we'll discuss connection-oriented service in detail and examine how it is implemented.

FOCUS ON SECURITY

SECURING TCP

Neither TCP nor UDP provide any encryption—the data that the sending process passes into its socket is the same data that travels over the network to the destination process. So, for example, if the sending process sends a password in cleartext (i.e., unencrypted) into its socket, the cleartext password will travel over all the links between sender and receiver, potentially getting sniffed and discovered at any of the intervening links. Because privacy and other security issues have become critical for many applications, the Internet community has developed an enhancement for TCP, called **Secure Sockets Layer (SSL)**.

TCP-enhanced-with-SSL not only does everything that traditional TCP does but also provides critical process-to-process security services, including encryption, data integrity, and end-point authentication. We emphasize that SSL is not a third Internet transport protocol, on the same level as TCP and UDP, but instead is an enhancement of TCP, with the enhancements being implemented in the application layer. In particular, if an application wants to use the services of SSL, it needs to include SSL code (existing, highly optimized libraries and classes) in both the client and server sides of the application. SSL has its own socket API that is similar to the traditional TCP socket API. When an application uses SSL, the sending process passes cleartext data to the SSL socket; SSL in the sending host then encrypts the data and passes the encrypted data to the TCP socket. The encrypted data travels over the Internet to the TCP socket in the receiving process. The receiving socket passes the encrypted data to SSL, which decrypts the data. Finally, SSL passes the cleartext data through its SSL socket to the receiving process. We'll cover SSL in some detail in Chapter 8.

- *Reliable data transfer service.* The communicating processes can rely on TCP to deliver all data sent without error and in the proper order. When one side of the application passes a stream of bytes into a socket, it can count on TCP to deliver the same stream of bytes to the receiving socket, with no missing or duplicate bytes.

TCP also includes a congestion-control mechanism, a service for the general welfare of the Internet rather than for the direct benefit of the communicating processes. The TCP congestion-control mechanism throttles a sending process (client or server) when the network is congested between sender and receiver. As we will see in Chapter 3, TCP congestion control also attempts to limit each TCP connection to its fair share of network bandwidth.

UDP Services

UDP is a no-frills, lightweight transport protocol, providing minimal services. UDP is connectionless, so there is no handshaking before the two processes start to communicate. UDP provides an unreliable data transfer service—that is, when a process sends a message into a UDP socket, UDP provides *no* guarantee that the message will ever reach the receiving process. Furthermore, messages that do arrive at the receiving process may arrive out of order.

UDP does not include a congestion-control mechanism, so the sending side of UDP can pump data into the layer below (the network layer) at any rate it pleases. (Note, however, that the actual end-to-end throughput may be less than this rate due to the limited transmission capacity of intervening links or due to congestion).

Services Not Provided by Internet Transport Protocols

We have organized transport protocol services along four dimensions: reliable data transfer, throughput, timing, and security. Which of these services are provided by TCP and UDP? We have already noted that TCP provides reliable end-to-end data transfer. And we also know that TCP can be easily enhanced at the application layer with SSL to provide security services. But in our brief description of TCP and UDP, conspicuously missing was any mention of throughput or timing guarantees—services *not* provided by today’s Internet transport protocols. Does this mean that time-sensitive applications such as Internet telephony cannot run in today’s Internet? The answer is clearly no—the Internet has been hosting time-sensitive applications for many years. These applications often work fairly well because they have been designed to cope, to the greatest extent possible, with this lack of guarantee. We’ll investigate several of these design tricks in Chapter 7. Nevertheless, clever design has its limitations when delay is excessive, or the end-to-end throughput is limited. In summary, today’s Internet can often provide satisfactory service to time-sensitive applications, but it cannot provide any timing or throughput guarantees.

Application	Application-Layer Protocol	Underlying Transport Protocol
Electronic mail	SMTP [RFC 5321]	TCP
Remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
File transfer	FTP [RFC 959]	TCP
Streaming multimedia	HTTP (e.g., YouTube)	TCP
Internet telephony	SIP [RFC 3261], RTP [RFC 3550], or proprietary (e.g., Skype)	UDP or TCP

Figure 2.5 ♦ Popular Internet applications, their application-layer protocols, and their underlying transport protocols

Figure 2.5 indicates the transport protocols used by some popular Internet applications. We see that e-mail, remote terminal access, the Web, and file transfer all use TCP. These applications have chosen TCP primarily because TCP provides reliable data transfer, guaranteeing that all data will eventually get to its destination. Because Internet telephony applications (such as Skype) can often tolerate some loss but require a minimal rate to be effective, developers of Internet telephony applications usually prefer to run their applications over UDP, thereby circumventing TCP's congestion control mechanism and packet overheads. But because many firewalls are configured to block (most types of) UDP traffic, Internet telephony applications often are designed to use TCP as a backup if UDP communication fails.

2.1.5 Application-Layer Protocols

We have just learned that network processes communicate with each other by sending messages into sockets. But how are these messages structured? What are the meanings of the various fields in the messages? When do the processes send the messages? These questions bring us into the realm of application-layer protocols. An **application-layer protocol** defines how an application's processes, running on different end systems, pass messages to each other. In particular, an application-layer protocol defines:

- The types of messages exchanged, for example, request messages and response messages
- The syntax of the various message types, such as the fields in the message and how the fields are delineated

- The semantics of the fields, that is, the meaning of the information in the fields
- Rules for determining when and how a process sends messages and responds to messages

Some application-layer protocols are specified in RFCs and are therefore in the public domain. For example, the Web's application-layer protocol, HTTP (the HyperText Transfer Protocol [RFC 2616]), is available as an RFC. If a browser developer follows the rules of the HTTP RFC, the browser will be able to retrieve Web pages from any Web server that has also followed the rules of the HTTP RFC. Many other application-layer protocols are proprietary and intentionally not available in the public domain. For example, Skype uses proprietary application-layer protocols.

It is important to distinguish between network applications and application-layer protocols. An application-layer protocol is only one piece of a network application (albeit, a very important piece of the application from our point of view!). Let's look at a couple of examples. The Web is a client-server application that allows users to obtain documents from Web servers on demand. The Web application consists of many components, including a standard for document formats (that is, HTML), Web browsers (for example, Firefox and Microsoft Internet Explorer), Web servers (for example, Apache and Microsoft servers), and an application-layer protocol. The Web's application-layer protocol, HTTP, defines the format and sequence of messages exchanged between browser and Web server. Thus, HTTP is only one piece (albeit, an important piece) of the Web application. As another example, an Internet e-mail application also has many components, including mail servers that house user mailboxes; mail clients (such as Microsoft Outlook) that allow users to read and create messages; a standard for defining the structure of an e-mail message; and application-layer protocols that define how messages are passed between servers, how messages are passed between servers and mail clients, and how the contents of message headers are to be interpreted. The principal application-layer protocol for electronic mail is SMTP (Simple Mail Transfer Protocol) [RFC 5321]. Thus, e-mail's principal application-layer protocol, SMTP, is only one piece (albeit, an important piece) of the e-mail application.

2.1.6 Network Applications Covered in This Book

New public domain and proprietary Internet applications are being developed every day. Rather than covering a large number of Internet applications in an encyclopedic manner, we have chosen to focus on a small number of applications that are both pervasive and important. In this chapter we discuss five important applications: the Web, file transfer, electronic mail, directory service, and P2P applications. We first discuss the Web, not only because it is an enormously popular application, but also because its application-layer protocol, HTTP, is straightforward and easy to understand. After covering the Web, we briefly examine FTP, because it provides a nice contrast to HTTP. We then discuss electronic mail, the Internet's first killer application. E-mail is more complex than the Web in the sense that it makes use of not one

but several application-layer protocols. After e-mail, we cover DNS, which provides a directory service for the Internet. Most users do not interact with DNS directly; instead, users invoke DNS indirectly through other applications (including the Web, file transfer, and electronic mail). DNS illustrates nicely how a piece of core network functionality (network-name to network-address translation) can be implemented at the application layer in the Internet. Finally, we discuss in this chapter several P2P applications, focusing on file sharing applications, and distributed lookup services. In Chapter 7, we'll cover multimedia applications, including streaming video and voice-over-IP.

2.2 The Web and HTTP

Until the early 1990s the Internet was used primarily by researchers, academics, and university students to log in to remote hosts, to transfer files from local hosts to remote hosts and vice versa, to receive and send news, and to receive and send electronic mail. Although these applications were (and continue to be) extremely useful, the Internet was essentially unknown outside of the academic and research communities. Then, in the early 1990s, a major new application arrived on the scene—the World Wide Web [Berners-Lee 1994]. The Web was the first Internet application that caught the general public's eye. It dramatically changed, and continues to change, how people interact inside and outside their work environments. It elevated the Internet from just one of many data networks to essentially the one and only data network.

Perhaps what appeals the most to users is that the Web operates *on demand*. Users receive what they want, when they want it. This is unlike traditional broadcast radio and television, which force users to tune in when the content provider makes the content available. In addition to being available on demand, the Web has many other wonderful features that people love and cherish. It is enormously easy for any individual to make information available over the Web—everyone can become a publisher at extremely low cost. Hyperlinks and search engines help us navigate through an ocean of Web sites. Graphics stimulate our senses. Forms, JavaScript, Java applets, and many other devices enable us to interact with pages and sites. And the Web serves as a platform for many killer applications emerging after 2003, including YouTube, Gmail, and Facebook.

2.2.1 Overview of HTTP

The **HyperText Transfer Protocol (HTTP)**, the Web's application-layer protocol, is at the heart of the Web. It is defined in [RFC 1945] and [RFC 2616]. HTTP is implemented in two programs: a client program and a server program. The client program and server program, executing on different end systems, talk to each other by exchanging HTTP messages. HTTP defines the structure of these messages and how the client and server exchange the messages. Before explaining HTTP in detail, we should review some Web terminology.

A **Web page** (also called a document) consists of objects. An **object** is simply a file—such as an HTML file, a JPEG image, a Java applet, or a video clip—that is addressable by a single URL. Most Web pages consist of a **base HTML file** and several referenced objects. For example, if a Web page contains HTML text and five JPEG images, then the Web page has six objects: the base HTML file plus the five images. The base HTML file references the other objects in the page with the objects’ URLs. Each URL has two components: the hostname of the server that houses the object and the object’s path name. For example, the URL

`http://www.someSchool.edu/someDepartment/picture.gif`

has `www.someSchool.edu` for a hostname and `/someDepartment/picture.gif` for a path name. Because **Web browsers** (such as Internet Explorer and Firefox) implement the client side of HTTP, in the context of the Web, we will use the words *browser* and *client* interchangeably. **Web servers**, which implement the server side of HTTP, house Web objects, each addressable by a URL. Popular Web servers include Apache and Microsoft Internet Information Server.

HTTP defines how Web clients request Web pages from Web servers and how servers transfer Web pages to clients. We discuss the interaction between client and server in detail later, but the general idea is illustrated in Figure 2.6. When a user requests a Web page (for example, clicks on a hyperlink), the browser sends HTTP request messages for the objects in the page to the server. The server receives the requests and responds with HTTP response messages that contain the objects.

HTTP uses TCP as its underlying transport protocol (rather than running on top of UDP). The HTTP client first initiates a TCP connection with the server. Once the connection is established, the browser and the server processes access TCP through their socket interfaces. As described in Section 2.1, on the client side the socket interface is the door between the client process and the TCP connection; on the server side it is the

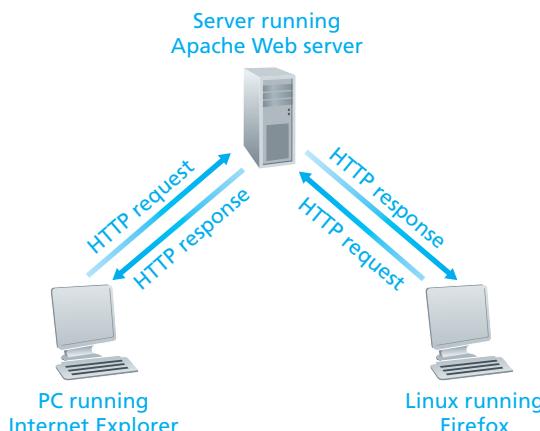


Figure 2.6 ♦ HTTP request-response behavior

door between the server process and the TCP connection. The client sends HTTP request messages into its socket interface and receives HTTP response messages from its socket interface. Similarly, the HTTP server receives request messages from its socket interface and sends response messages into its socket interface. Once the client sends a message into its socket interface, the message is out of the client's hands and is "in the hands" of TCP. Recall from Section 2.1 that TCP provides a reliable data transfer service to HTTP. This implies that each HTTP request message sent by a client process eventually arrives intact at the server; similarly, each HTTP response message sent by the server process eventually arrives intact at the client. Here we see one of the great advantages of a layered architecture—HTTP need not worry about lost data or the details of how TCP recovers from loss or reordering of data within the network. That is the job of TCP and the protocols in the lower layers of the protocol stack.

It is important to note that the server sends requested files to clients without storing any state information about the client. If a particular client asks for the same object twice in a period of a few seconds, the server does not respond by saying that it just served the object to the client; instead, the server resends the object, as it has completely forgotten what it did earlier. Because an HTTP server maintains no information about the clients, HTTP is said to be a **stateless protocol**. We also remark that the Web uses the client-server application architecture, as described in Section 2.1. A Web server is always on, with a fixed IP address, and it services requests from potentially millions of different browsers.

2.2.2 Non-Persistent and Persistent Connections

In many Internet applications, the client and server communicate for an extended period of time, with the client making a series of requests and the server responding to each of the requests. Depending on the application and on how the application is being used, the series of requests may be made back-to-back, periodically at regular intervals, or intermittently. When this client-server interaction is taking place over TCP, the application developer needs to make an important decision—should each request/response pair be sent over a *separate* TCP connection, or should all of the requests and their corresponding responses be sent over the *same* TCP connection? In the former approach, the application is said to use **non-persistent connections**; and in the latter approach, **persistent connections**. To gain a deep understanding of this design issue, let's examine the advantages and disadvantages of persistent connections in the context of a specific application, namely, HTTP, which can use both non-persistent connections and persistent connections. Although HTTP uses persistent connections in its default mode, HTTP clients and servers can be configured to use non-persistent connections instead.

HTTP with Non-Persistent Connections

Let's walk through the steps of transferring a Web page from server to client for the case of non-persistent connections. Let's suppose the page consists of a base HTML

file and 10 JPEG images, and that all 11 of these objects reside on the same server. Further suppose the URL for the base HTML file is

`http://www.someSchool.edu/someDepartment/home.index`

Here is what happens:

1. The HTTP client process initiates a TCP connection to the server `www.someSchool.edu` on port number 80, which is the default port number for HTTP. Associated with the TCP connection, there will be a socket at the client and a socket at the server.
2. The HTTP client sends an HTTP request message to the server via its socket. The request message includes the path name `/someDepartment/home.index`. (We will discuss HTTP messages in some detail below.)
3. The HTTP server process receives the request message via its socket, retrieves the object `/someDepartment/home.index` from its storage (RAM or disk), encapsulates the object in an HTTP response message, and sends the response message to the client via its socket.
4. The HTTP server process tells TCP to close the TCP connection. (But TCP doesn't actually terminate the connection until it knows for sure that the client has received the response message intact.)
5. The HTTP client receives the response message. The TCP connection terminates. The message indicates that the encapsulated object is an HTML file. The client extracts the file from the response message, examines the HTML file, and finds references to the 10 JPEG objects.
6. The first four steps are then repeated for each of the referenced JPEG objects.

As the browser receives the Web page, it displays the page to the user. Two different browsers may interpret (that is, display to the user) a Web page in somewhat different ways. HTTP has nothing to do with how a Web page is interpreted by a client. The HTTP specifications ([RFC 1945] and [RFC 2616]) define only the communication protocol between the client HTTP program and the server HTTP program.

The steps above illustrate the use of non-persistent connections, where each TCP connection is closed after the server sends the object—the connection does not persist for other objects. Note that each TCP connection transports exactly one request message and one response message. Thus, in this example, when a user requests the Web page, 11 TCP connections are generated.

In the steps described above, we were intentionally vague about whether the client obtains the 10 JPEGs over 10 serial TCP connections, or whether some of the JPEGs are obtained over parallel TCP connections. Indeed, users can configure modern browsers to control the degree of parallelism. In their default modes, most browsers open 5 to 10 parallel TCP connections, and each of these connections handles one request-response transaction. If the user prefers, the maximum number of

parallel connections can be set to one, in which case the 10 connections are established serially. As we'll see in the next chapter, the use of parallel connections shortens the response time.

Before continuing, let's do a back-of-the-envelope calculation to estimate the amount of time that elapses from when a client requests the base HTML file until the entire file is received by the client. To this end, we define the **round-trip time (RTT)**, which is the time it takes for a small packet to travel from client to server and then back to the client. The RTT includes packet-propagation delays, packet-queuing delays in intermediate routers and switches, and packet-processing delays. (These delays were discussed in Section 1.4.) Now consider what happens when a user clicks on a hyperlink. As shown in Figure 2.7, this causes the browser to initiate a TCP connection between the browser and the Web server; this involves a "three-way handshake"—the client sends a small TCP segment to the server, the server acknowledges and responds with a small TCP segment, and, finally, the client acknowledges back to the server. The first two parts of the three-way handshake take one RTT. After completing the first two parts of the handshake, the client sends the HTTP request message combined with the third part of

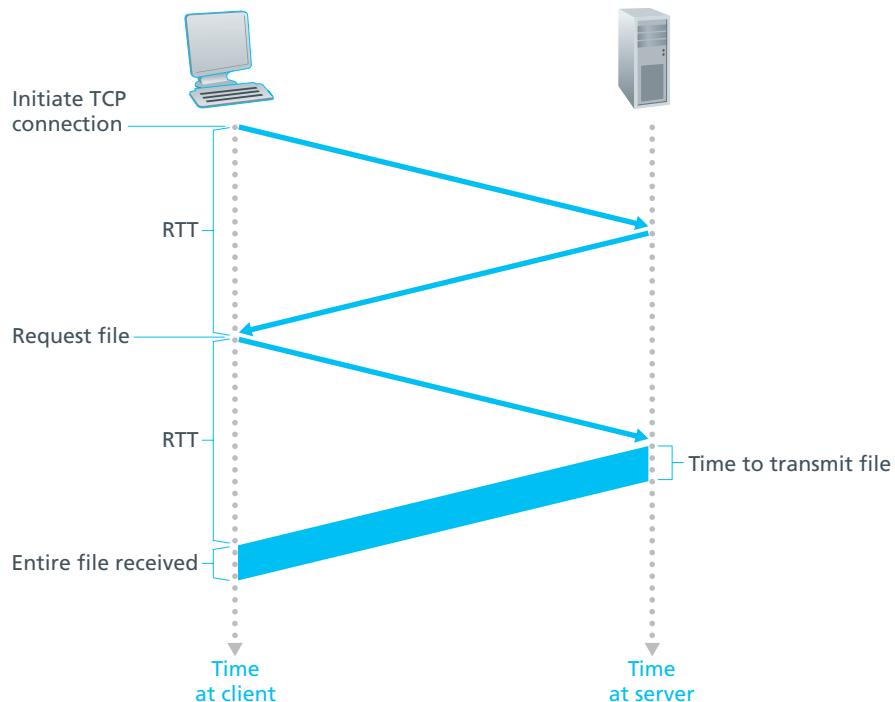


Figure 2.7 ♦ Back-of-the-envelope calculation for the time needed to request and receive an HTML file

the three-way handshake (the acknowledgment) into the TCP connection. Once the request message arrives at the server, the server sends the HTML file into the TCP connection. This HTTP request/response eats up another RTT. Thus, roughly, the total response time is two RTTs plus the transmission time at the server of the HTML file.

HTTP with Persistent Connections

Non-persistent connections have some shortcomings. First, a brand-new connection must be established and maintained for *each requested object*. For each of these connections, TCP buffers must be allocated and TCP variables must be kept in both the client and server. This can place a significant burden on the Web server, which may be serving requests from hundreds of different clients simultaneously. Second, as we just described, each object suffers a delivery delay of two RTTs—one RTT to establish the TCP connection and one RTT to request and receive an object.

With persistent connections, the server leaves the TCP connection open after sending a response. Subsequent requests and responses between the same client and server can be sent over the same connection. In particular, an entire Web page (in the example above, the base HTML file and the 10 images) can be sent over a single persistent TCP connection. Moreover, multiple Web pages residing on the same server can be sent from the server to the same client over a single persistent TCP connection. These requests for objects can be made back-to-back, without waiting for replies to pending requests (pipelining). Typically, the HTTP server closes a connection when it isn't used for a certain time (a configurable timeout interval). When the server receives the back-to-back requests, it sends the objects back-to-back. The default mode of HTTP uses persistent connections with pipelining. We'll quantitatively compare the performance of non-persistent and persistent connections in the homework problems of Chapters 2 and 3. You are also encouraged to see [Heidemann 1997; Nielsen 1997].

2.2.3 HTTP Message Format

The HTTP specifications [RFC 1945; RFC 2616] include the definitions of the HTTP message formats. There are two types of HTTP messages, request messages and response messages, both of which are discussed below.

HTTP Request Message

Below we provide a typical HTTP request message:

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
```

```
Connection: close
User-agent: Mozilla/5.0
Accept-language: fr
```

We can learn a lot by taking a close look at this simple request message. First of all, we see that the message is written in ordinary ASCII text, so that your ordinary computer-literate human being can read it. Second, we see that the message consists of five lines, each followed by a carriage return and a line feed. The last line is followed by an additional carriage return and line feed. Although this particular request message has five lines, a request message can have many more lines or as few as one line. The first line of an HTTP request message is called the **request line**; the subsequent lines are called the **header lines**. The request line has three fields: the method field, the URL field, and the HTTP version field. The method field can take on several different values, including **GET**, **POST**, **HEAD**, **PUT**, and **DELETE**. The great majority of HTTP request messages use the **GET** method. The **GET** method is used when the browser requests an object, with the requested object identified in the URL field. In this example, the browser is requesting the object **/somedir/page.html**. The version is self-explanatory; in this example, the browser implements version HTTP/1.1.

Now let's look at the header lines in the example. The header line **Host: www.someschool.edu** specifies the host on which the object resides. You might think that this header line is unnecessary, as there is already a TCP connection in place to the host. But, as we'll see in Section 2.2.5, the information provided by the host header line is required by Web proxy caches. By including the **Connection: close** header line, the browser is telling the server that it doesn't want to bother with persistent connections; it wants the server to close the connection after sending the requested object. The **User-agent:** header line specifies the user agent, that is, the browser type that is making the request to the server. Here the user agent is Mozilla/5.0, a Firefox browser. This header line is useful because the server can actually send different versions of the same object to different types of user agents. (Each of the versions is addressed by the same URL.) Finally, the **Accept-language:** header indicates that the user prefers to receive a French version of the object, if such an object exists on the server; otherwise, the server should send its default version. The **Accept-language:** header is just one of many content negotiation headers available in HTTP.

Having looked at an example, let's now look at the general format of a request message, as shown in Figure 2.8. We see that the general format closely follows our earlier example. You may have noticed, however, that after the header lines (and the additional carriage return and line feed) there is an “entity body.” The entity body is empty with the **GET** method, but is used with the **POST** method. An HTTP client often uses the **POST** method when the user fills out a form—for example, when a user provides search words to a search engine. With a **POST** message, the user is still requesting a Web page from the server, but the specific contents of the Web page

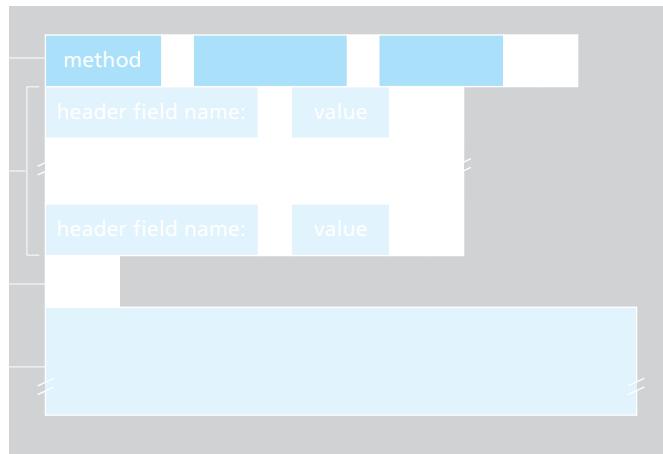


Figure 2.8 ♦ General format of an HTTP request message

depend on what the user entered into the form fields. If the value of the method field is POST, then the entity body contains what the user entered into the form fields.

We would be remiss if we didn't mention that a request generated with a form does not necessarily use the POST method. Instead, HTML forms often use the GET method and include the inputted data (in the form fields) in the requested URL. For example, if a form uses the GET method, has two fields, and the inputs to the two fields are `monkeys` and `bananas`, then the URL will have the structure `www.somesite.com/animalsearch?monkeys&bananas`. In your day-to-day Web surfing, you have probably noticed extended URLs of this sort.

The HEAD method is similar to the GET method. When a server receives a request with the HEAD method, it responds with an HTTP message but it leaves out the requested object. Application developers often use the HEAD method for debugging. The PUT method is often used in conjunction with Web publishing tools. It allows a user to upload an object to a specific path (directory) on a specific Web server. The PUT method is also used by applications that need to upload objects to Web servers. The DELETE method allows a user, or an application, to delete an object on a Web server.

HTTP Response Message

Below we provide a typical HTTP response message. This response message could be the response to the example request message just discussed.

```
HTTP/1.1 200 OK
Connection: close
```

```
Date: Tue, 09 Aug 2011 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Tue, 09 Aug 2011 15:11:03 GMT
Content-Length: 6821
Content-Type: text/html

(data data data data data ...)
```

Let's take a careful look at this response message. It has three sections: an initial **status line**, six **header lines**, and then the **entity body**. The entity body is the meat of the message—it contains the requested object itself (represented by `data data data data data ...`). The status line has three fields: the protocol version field, a status code, and a corresponding status message. In this example, the status line indicates that the server is using HTTP/1.1 and that everything is OK (that is, the server has found, and is sending, the requested object).

Now let's look at the header lines. The server uses the `Connection: close` header line to tell the client that it is going to close the TCP connection after sending the message. The `Date:` header line indicates the time and date when the HTTP response was created and sent by the server. Note that this is not the time when the object was created or last modified; it is the time when the server retrieves the object from its file system, inserts the object into the response message, and sends the response message. The `Server:` header line indicates that the message was generated by an Apache Web server; it is analogous to the `User-agent:` header line in the HTTP request message. The `Last-Modified:` header line indicates the time and date when the object was created or last modified. The `Last-Modified:` header, which we will soon cover in more detail, is critical for object caching, both in the local client and in network cache servers (also known as proxy servers). The `Content-Length:` header line indicates the number of bytes in the object being sent. The `Content-Type:` header line indicates that the object in the entity body is HTML text. (The object type is officially indicated by the `Content-Type:` header and not by the file extension.)

Having looked at an example, let's now examine the general format of a response message, which is shown in Figure 2.9. This general format of the response message matches the previous example of a response message. Let's say a few additional words about status codes and their phrases. The status code and associated phrase indicate the result of the request. Some common status codes and associated phrases include:

- **200 OK:** Request succeeded and the information is returned in the response.
- **301 Moved Permanently:** Requested object has been permanently moved; the new URL is specified in `Location:` header of the response message. The client software will automatically retrieve the new URL.

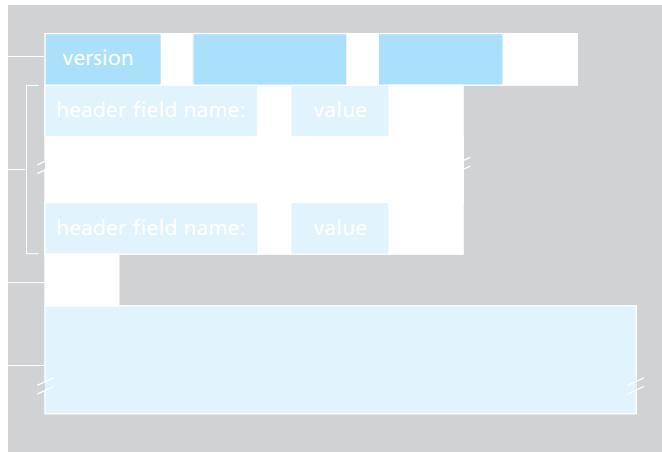


Figure 2.9 ♦ General format of an HTTP response message

- **400 Bad Request:** This is a generic error code indicating that the request could not be understood by the server.
- **404 Not Found:** The requested document does not exist on this server.
- **505 HTTP Version Not Supported:** The requested HTTP protocol version is not supported by the server.

How would you like to see a real HTTP response message? This is highly recommended and very easy to do! First Telnet into your favorite Web server. Then type in a one-line request message for some object that is housed on the server. For example, if you have access to a command prompt, type:

```
telnet cis.poly.edu 80
```

```
GET /~ross/ HTTP/1.1  
Host: cis.poly.edu
```

(Press the carriage return twice after typing the last line.) This opens a TCP connection to port 80 of the host `cis.poly.edu` and then sends the HTTP request message. You should see a response message that includes the base HTML file of Professor Ross's homepage. If you'd rather just see the HTTP message lines and not receive the object itself, replace `GET` with `HEAD`. Finally, replace `/~ross/` with `/~banana/` and see what kind of response message you get.

In this section we discussed a number of header lines that can be used within HTTP request and response messages. The HTTP specification defines many, many



more header lines that can be inserted by browsers, Web servers, and network cache servers. We have covered only a small number of the totality of header lines. We'll cover a few more below and another small number when we discuss network Web caching in Section 2.2.5. A highly readable and comprehensive discussion of the HTTP protocol, including its headers and status codes, is given in [Krishnamurthy 2001].

How does a browser decide which header lines to include in a request message? How does a Web server decide which header lines to include in a response message? A browser will generate header lines as a function of the browser type and version (for example, an HTTP/1.0 browser will not generate any 1.1 header lines), the user configuration of the browser (for example, preferred language), and whether the browser currently has a cached, but possibly out-of-date, version of the object. Web servers behave similarly: There are different products, versions, and configurations, all of which influence which header lines are included in response messages.

2.2.4 User-Server Interaction: Cookies

We mentioned above that an HTTP server is stateless. This simplifies server design and has permitted engineers to develop high-performance Web servers that can handle thousands of simultaneous TCP connections. However, it is often desirable for a Web site to identify users, either because the server wishes to restrict user access or because it wants to serve content as a function of the user identity. For these purposes, HTTP uses cookies. Cookies, defined in [RFC 6265], allow sites to keep track of users. Most major commercial Web sites use cookies today.

As shown in Figure 2.10, cookie technology has four components: (1) a cookie header line in the HTTP response message; (2) a cookie header line in the HTTP request message; (3) a cookie file kept on the user's end system and managed by the user's browser; and (4) a back-end database at the Web site. Using Figure 2.10, let's walk through an example of how cookies work. Suppose Susan, who always accesses the Web using Internet Explorer from her home PC, contacts Amazon.com for the first time. Let us suppose that in the past she has already visited the eBay site. When the request comes into the Amazon Web server, the server creates a unique identification number and creates an entry in its back-end database that is indexed by the identification number. The Amazon Web server then responds to Susan's browser, including in the HTTP response a **Set-cookie:** header, which contains the identification number. For example, the header line might be:

```
Set-cookie: 1678
```

When Susan's browser receives the HTTP response message, it sees the **Set-cookie:** header. The browser then appends a line to the special cookie file that it manages. This line includes the hostname of the server and the identification number in the **Set-cookie:** header. Note that the cookie file already has an entry for

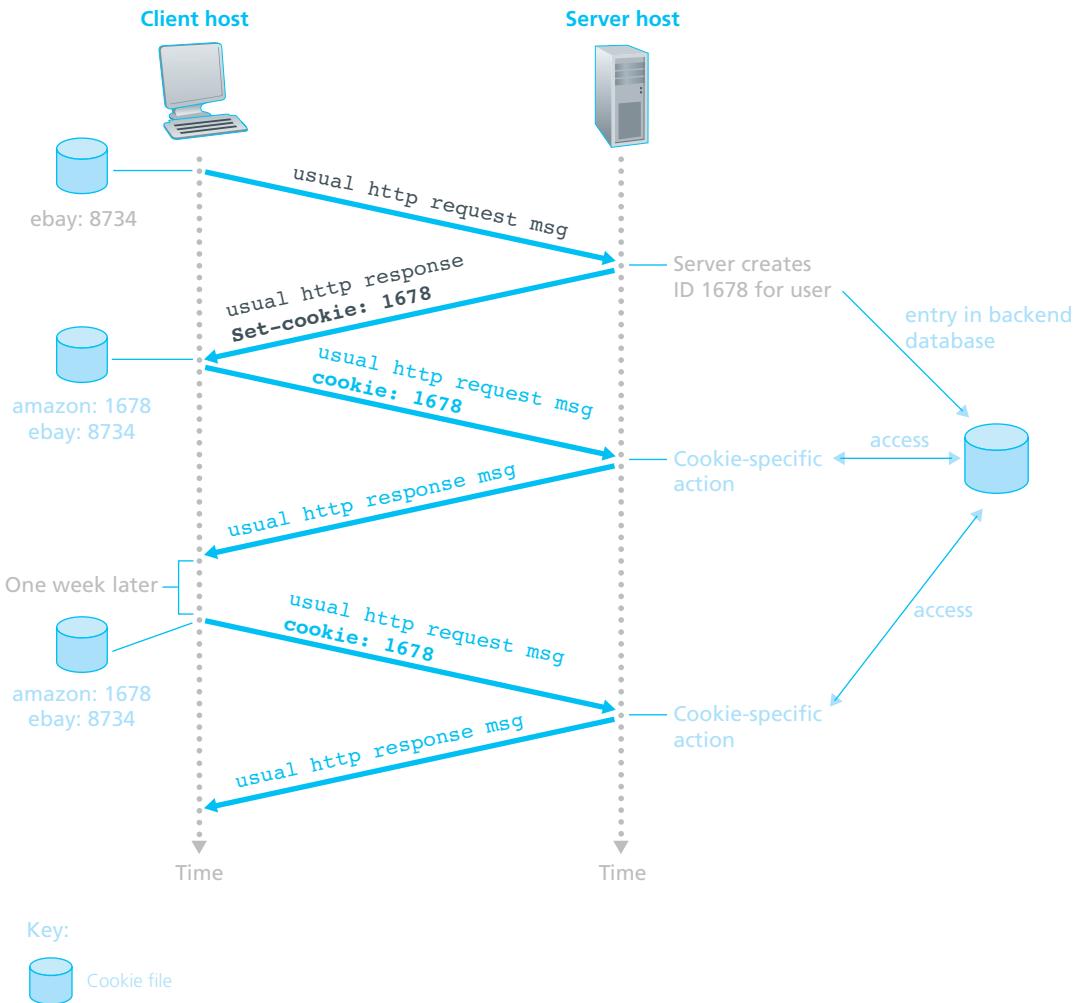


Figure 2.10 ♦ Keeping user state with cookies

eBay, since Susan has visited that site in the past. As Susan continues to browse the Amazon site, each time she requests a Web page, her browser consults her cookie file, extracts her identification number for this site, and puts a cookie header line that includes the identification number in the HTTP request. Specifically, each of her HTTP requests to the Amazon server includes the header line:

Cookie: 1678

In this manner, the Amazon server is able to track Susan's activity at the Amazon site. Although the Amazon Web site does not necessarily know Susan's name, it knows exactly which pages user 1678 visited, in which order, and at what times! Amazon uses cookies to provide its shopping cart service—Amazon can maintain a list of all of Susan's intended purchases, so that she can pay for them collectively at the end of the session.

If Susan returns to Amazon's site, say, one week later, her browser will continue to put the header line `Cookie: 1678` in the request messages. Amazon also recommends products to Susan based on Web pages she has visited at Amazon in the past. If Susan also registers herself with Amazon—providing full name, e-mail address, postal address, and credit card information—Amazon can then include this information in its database, thereby associating Susan's name with her identification number (and all of the pages she has visited at the site in the past!). This is how Amazon and other e-commerce sites provide “one-click shopping”—when Susan chooses to purchase an item during a subsequent visit, she doesn't need to re-enter her name, credit card number, or address.

From this discussion we see that cookies can be used to identify a user. The first time a user visits a site, the user can provide a user identification (possibly his or her name). During the subsequent sessions, the browser passes a cookie header to the server, thereby identifying the user to the server. Cookies can thus be used to create a user session layer on top of stateless HTTP. For example, when a user logs in to a Web-based e-mail application (such as Hotmail), the browser sends cookie information to the server, permitting the server to identify the user throughout the user's session with the application.

Although cookies often simplify the Internet shopping experience for the user, they are controversial because they can also be considered as an invasion of privacy. As we just saw, using a combination of cookies and user-supplied account information, a Web site can learn a lot about a user and potentially sell this information to a third party. Cookie Central [Cookie Central 2012] includes extensive information on the cookie controversy.

2.2.5 Web Caching

A **Web cache**—also called a **proxy server**—is a network entity that satisfies HTTP requests on the behalf of an origin Web server. The Web cache has its own disk storage and keeps copies of recently requested objects in this storage. As shown in Figure 2.11, a user's browser can be configured so that all of the user's HTTP requests are first directed to the Web cache. Once a browser is configured, each browser request for an object is first directed to the Web cache. As an example, suppose a browser is requesting the object `http://www.someschool.edu/campus.gif`. Here is what happens:

1. The browser establishes a TCP connection to the Web cache and sends an HTTP request for the object to the Web cache.

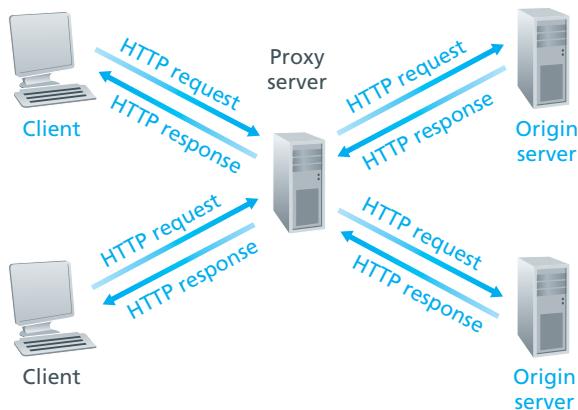


Figure 2.11 ♦ Clients requesting objects through a Web cache

2. The Web cache checks to see if it has a copy of the object stored locally. If it does, the Web cache returns the object within an HTTP response message to the client browser.
3. If the Web cache does not have the object, the Web cache opens a TCP connection to the origin server, that is, to `www.someschool.edu`. The Web cache then sends an HTTP request for the object into the cache-to-server TCP connection. After receiving this request, the origin server sends the object within an HTTP response to the Web cache.
4. When the Web cache receives the object, it stores a copy in its local storage and sends a copy, within an HTTP response message, to the client browser (over the existing TCP connection between the client browser and the Web cache).

Note that a cache is both a server and a client at the same time. When it receives requests from and sends responses to a browser, it is a server. When it sends requests to and receives responses from an origin server, it is a client.

Typically a Web cache is purchased and installed by an ISP. For example, a university might install a cache on its campus network and configure all of the campus browsers to point to the cache. Or a major residential ISP (such as AOL) might install one or more caches in its network and preconfigure its shipped browsers to point to the installed caches.

Web caching has seen deployment in the Internet for two reasons. First, a Web cache can substantially reduce the response time for a client request, particularly if the bottleneck bandwidth between the client and the origin server is much less than the bottleneck bandwidth between the client and the cache. If there is a high-speed connection between the client and the cache, as there often is, and if the cache has the requested object, then the cache will be able to deliver the object rapidly to the client. Second, as we will soon illustrate with an example, Web caches can substantially reduce traffic on

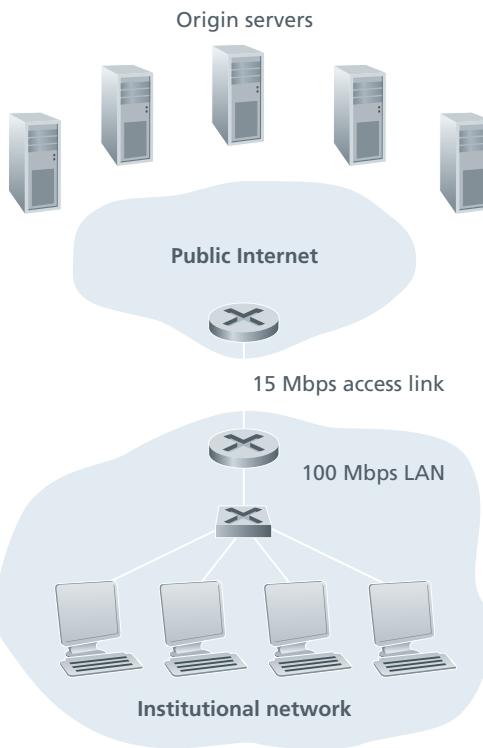


Figure 2.12 ♦ Bottleneck between an institutional network and the Internet

an institution's access link to the Internet. By reducing traffic, the institution (for example, a company or a university) does not have to upgrade bandwidth as quickly, thereby reducing costs. Furthermore, Web caches can substantially reduce Web traffic in the Internet as a whole, thereby improving performance for all applications.

To gain a deeper understanding of the benefits of caches, let's consider an example in the context of Figure 2.12. This figure shows two networks—the institutional network and the rest of the public Internet. The institutional network is a high-speed LAN. A router in the institutional network and a router in the Internet are connected by a 15 Mbps link. The origin servers are attached to the Internet but are located all over the globe. Suppose that the average object size is 1 Mbits and that the average request rate from the institution's browsers to the origin servers is 15 requests per second. Suppose that the HTTP request messages are negligibly small and thus create no traffic in the networks or in the access link (from institutional router to Internet router). Also suppose that the amount of time it takes from when the router on the Internet side of the access link in Figure 2.12 forwards an HTTP request (within an IP datagram) until it receives the response (typically within many IP datagrams) is two seconds on average. Informally, we refer to this last delay as the “Internet delay.”

The total response time—that is, the time from the browser’s request of an object until its receipt of the object—is the sum of the LAN delay, the access delay (that is, the delay between the two routers), and the Internet delay. Let’s now do a very crude calculation to estimate this delay. The traffic intensity on the LAN (see Section 1.4.2) is

$$(15 \text{ requests/sec}) \cdot (1 \text{ Mbits/request})/(100 \text{ Mbps}) = 0.15$$

whereas the traffic intensity on the access link (from the Internet router to institution router) is

$$(15 \text{ requests/sec}) \cdot (1 \text{ Mbits/request})/(15 \text{ Mbps}) = 1$$

A traffic intensity of 0.15 on a LAN typically results in, at most, tens of milliseconds of delay; hence, we can neglect the LAN delay. However, as discussed in Section 1.4.2, as the traffic intensity approaches 1 (as is the case of the access link in Figure 2.12), the delay on a link becomes very large and grows without bound. Thus, the average response time to satisfy requests is going to be on the order of minutes, if not more, which is unacceptable for the institution’s users. Clearly something must be done.

One possible solution is to increase the access rate from 15 Mbps to, say, 100 Mbps. This will lower the traffic intensity on the access link to 0.15, which translates to negligible delays between the two routers. In this case, the total response time will roughly be two seconds, that is, the Internet delay. But this solution also means that the institution must upgrade its access link from 15 Mbps to 100 Mbps, a costly proposition.

Now consider the alternative solution of not upgrading the access link but instead installing a Web cache in the institutional network. This solution is illustrated in Figure 2.13. Hit rates—the fraction of requests that are satisfied by a cache—typically range from 0.2 to 0.7 in practice. For illustrative purposes, let’s suppose that the cache provides a hit rate of 0.4 for this institution. Because the clients and the cache are connected to the same high-speed LAN, 40 percent of the requests will be satisfied almost immediately, say, within 10 milliseconds, by the cache. Nevertheless, the remaining 60 percent of the requests still need to be satisfied by the origin servers. But with only 60 percent of the requested objects passing through the access link, the traffic intensity on the access link is reduced from 1.0 to 0.6. Typically, a traffic intensity less than 0.8 corresponds to a small delay, say, tens of milliseconds, on a 15 Mbps link. This delay is negligible compared with the two-second Internet delay. Given these considerations, average delay therefore is

$$0.4 \cdot (0.01 \text{ seconds}) + 0.6 \cdot (2.01 \text{ seconds})$$

which is just slightly greater than 1.2 seconds. Thus, this second solution provides an even lower response time than the first solution, and it doesn’t require the institution to upgrade its link to the Internet. The institution does, of course, have to purchase

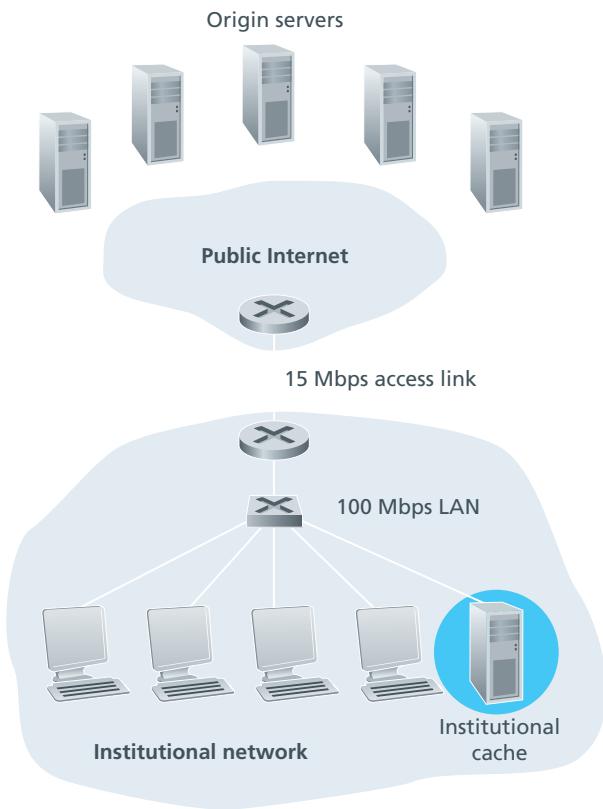


Figure 2.13 ♦ Adding a cache to the institutional network

and install a Web cache. But this cost is low—many caches use public-domain software that runs on inexpensive PCs.

Through the use of **Content Distribution Networks (CDNs)**, Web caches are increasingly playing an important role in the Internet. A CDN company installs many geographically distributed caches throughout the Internet, thereby localizing much of the traffic. There are shared CDNs (such as Akamai and Limelight) and dedicated CDNs (such as Google and Microsoft). We will discuss CDNs in more detail in Chapter 7.

2.2.6 The Conditional GET

Although caching can reduce user-perceived response times, it introduces a new problem—the copy of an object residing in the cache may be stale. In other words, the object housed in the Web server may have been modified since the copy was cached at the client. Fortunately, HTTP has a mechanism that allows a cache to verify that its objects are up to date. This mechanism is called the **conditional GET**. An HTTP

request message is a so-called conditional GET message if (1) the request message uses the `GET` method and (2) the request message includes an `If-Modified-Since:` header line.

To illustrate how the conditional GET operates, let's walk through an example. First, on the behalf of a requesting browser, a proxy cache sends a request message to a Web server:

```
GET /fruit/kiwi.gif HTTP/1.1
Host: www.exotiquecuisine.com
```

Second, the Web server sends a response message with the requested object to the cache:

```
HTTP/1.1 200 OK
Date: Sat, 8 Oct 2011 15:39:29
Server: Apache/1.3.0 (Unix)
Last-Modified: Wed, 7 Sep 2011 09:23:24
Content-Type: image/gif

(data data data data data ...)
```

The cache forwards the object to the requesting browser but also caches the object locally. Importantly, the cache also stores the last-modified date along with the object. Third, one week later, another browser requests the same object via the cache, and the object is still in the cache. Since this object may have been modified at the Web server in the past week, the cache performs an up-to-date check by issuing a conditional GET. Specifically, the cache sends:

```
GET /fruit/kiwi.gif HTTP/1.1
Host: www.exotiquecuisine.com
If-modified-since: Wed, 7 Sep 2011 09:23:24
```

Note that the value of the `If-modified-since:` header line is exactly equal to the value of the `Last-Modified:` header line that was sent by the server one week ago. This conditional GET is telling the server to send the object only if the object has been modified since the specified date. Suppose the object has not been modified since 7 Sep 2011 09:23:24. Then, fourth, the Web server sends a response message to the cache:

```
HTTP/1.1 304 Not Modified
Date: Sat, 15 Oct 2011 15:39:29
Server: Apache/1.3.0 (Unix)

(empty entity body)
```

We see that in response to the conditional GET, the Web server still sends a response message but does not include the requested object in the response message. Including the requested object would only waste bandwidth and increase user-perceived response time, particularly if the object is large. Note that this last response message has 304 Not Modified in the status line, which tells the cache that it can go ahead and forward its (the proxy cache's) cached copy of the object to the requesting browser.

This ends our discussion of HTTP, the first Internet protocol (an application-layer protocol) that we've studied in detail. We've seen the format of HTTP messages and the actions taken by the Web client and server as these messages are sent and received. We've also studied a bit of the Web's application infrastructure, including caches, cookies, and back-end databases, all of which are tied in some way to the HTTP protocol.

2.3 File Transfer: FTP

In a typical FTP session, the user is sitting in front of one host (the local host) and wants to transfer files to or from a remote host. In order for the user to access the remote account, the user must provide a user identification and a password. After providing this authorization information, the user can transfer files from the local file system to the remote file system and vice versa. As shown in Figure 2.14, the user interacts with FTP through an FTP user agent. The user first provides the hostname of the remote host, causing the FTP client process in the local host to establish a TCP connection with the FTP server process in the remote host. The user then provides the user identification and password, which are sent over the TCP connection as part of FTP commands. Once the server has authorized the user, the user copies one or more files stored in the local file system into the remote file system (or vice versa).

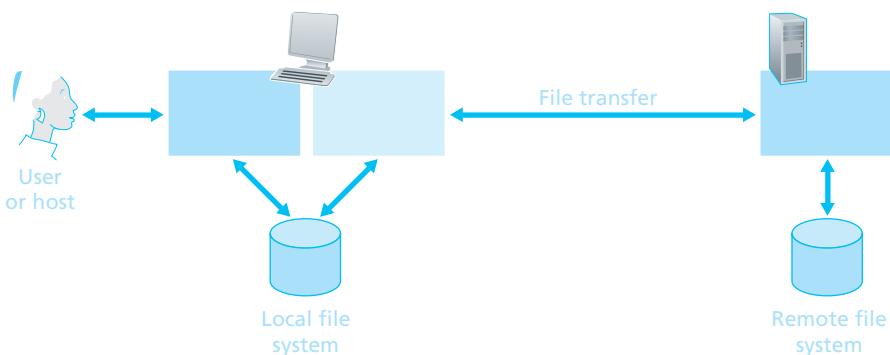


Figure 2.14 ♦ FTP moves files between local and remote file systems



Figure 2.15 ♦ Control and data connections

HTTP and FTP are both file transfer protocols and have many common characteristics; for example, they both run on top of TCP. However, the two application-layer protocols have some important differences. The most striking difference is that FTP uses two parallel TCP connections to transfer a file, a **control connection** and a **data connection**. The control connection is used for sending control information between the two hosts—information such as user identification, password, commands to change remote directory, and commands to “put” and “get” files. The data connection is used to actually send a file. Because FTP uses a separate control connection, FTP is said to send its control information **out-of-band**. HTTP, as you recall, sends request and response header lines into the same TCP connection that carries the transferred file itself. For this reason, HTTP is said to send its control information **in-band**. In the next section, we’ll see that SMTP, the main protocol for electronic mail, also sends control information in-band. The FTP control and data connections are illustrated in Figure 2.15.

When a user starts an FTP session with a remote host, the client side of FTP (user) first initiates a control TCP connection with the server side (remote host) on server port number 21. The client side of FTP sends the user identification and password over this control connection. The client side of FTP also sends, over the control connection, commands to change the remote directory. When the server side receives a command for a file transfer over the control connection (either to, or from, the remote host), the server side initiates a TCP data connection to the client side. FTP sends exactly one file over the data connection and then closes the data connection. If, during the same session, the user wants to transfer another file, FTP opens another data connection. Thus, with FTP, the control connection remains open throughout the duration of the user session, but a new data connection is created for each file transferred within a session (that is, the data connections are non-persistent).

Throughout a session, the FTP server must maintain **state** about the user. In particular, the server must associate the control connection with a specific user account, and the server must keep track of the user’s current directory as the user wanders about the remote directory tree. Keeping track of this state information for each ongoing user session significantly constrains the total number of sessions that FTP can maintain simultaneously. Recall that HTTP, on the other hand, is stateless—it does not have to keep track of any user state.

2.3.1 FTP Commands and Replies

We end this section with a brief discussion of some of the more common FTP commands and replies. The commands, from client to server, and replies, from server to client, are sent across the control connection in 7-bit ASCII format. Thus, like HTTP commands, FTP commands are readable by people. In order to delineate successive commands, a carriage return and line feed end each command. Each command consists of four uppercase ASCII characters, some with optional arguments. Some of the more common commands are given below:

- **USER username:** Used to send the user identification to the server.
- **PASS password:** Used to send the user password to the server.
- **LIST:** Used to ask the server to send back a list of all the files in the current remote directory. The list of files is sent over a (new and non-persistent) data connection rather than the control TCP connection.
- **RETR filename:** Used to retrieve (that is, get) a file from the current directory of the remote host. This command causes the remote host to initiate a data connection and to send the requested file over the data connection.
- **STOR filename:** Used to store (that is, put) a file into the current directory of the remote host.

There is typically a one-to-one correspondence between the command that the user issues and the FTP command sent across the control connection. Each command is followed by a reply, sent from server to client. The replies are three-digit numbers, with an optional message following the number. This is similar in structure to the status code and phrase in the status line of the HTTP response message. Some typical replies, along with their possible messages, are as follows:

- 331 Username OK, password required
- 125 Data connection already open; transfer starting
- 425 Can't open data connection
- 452 Error writing file

Readers who are interested in learning about the other FTP commands and replies are encouraged to read RFC 959.

2.4 Electronic Mail in the Internet

Electronic mail has been around since the beginning of the Internet. It was the most popular application when the Internet was in its infancy [Segaller 1998], and has

become more and more elaborate and powerful over the years. It remains one of the Internet's most important and utilized applications.

As with ordinary postal mail, e-mail is an asynchronous communication medium—people send and read messages when it is convenient for them, without having to coordinate with other people's schedules. In contrast with postal mail, electronic mail is fast, easy to distribute, and inexpensive. Modern e-mail has many powerful features, including messages with attachments, hyperlinks, HTML-formatted text, and embedded photos.

In this section, we examine the application-layer protocols that are at the heart of Internet e-mail. But before we jump into an in-depth discussion of these protocols, let's take a high-level view of the Internet mail system and its key components.

Figure 2.16 presents a high-level view of the Internet mail system. We see from this diagram that it has three major components: **user agents**, **mail servers**, and the

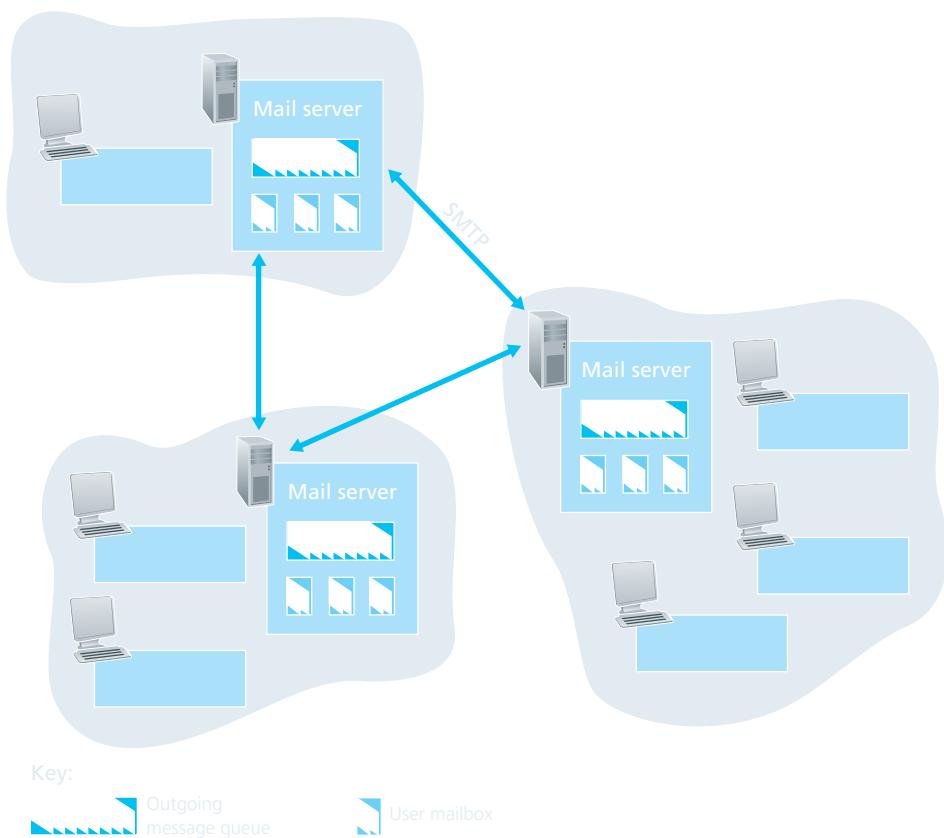


Figure 2.16 ♦ A high-level view of the Internet e-mail system

Simple Mail Transfer Protocol (SMTP). We now describe each of these components in the context of a sender, Alice, sending an e-mail message to a recipient, Bob. User agents allow users to read, reply to, forward, save, and compose messages. Microsoft Outlook and Apple Mail are examples of user agents for e-mail. When Alice is finished composing her message, her user agent sends the message to her mail server, where the message is placed in the mail server’s outgoing message queue. When Bob wants to read a message, his user agent retrieves the message from his mailbox in his mail server.

Mail servers form the core of the e-mail infrastructure. Each recipient, such as Bob, has a **mailbox** located in one of the mail servers. Bob’s mailbox manages and maintains the messages that have been sent to him. A typical message starts its journey in the sender’s user agent, travels to the sender’s mail server, and travels to the recipient’s mail server, where it is deposited in the recipient’s mailbox.

CASE HISTORY

WEB E-MAIL

In December 1995, just a few years after the Web was “invented,” Sabeer Bhatia and Jack Smith visited the Internet venture capitalist Draper Fisher Jurvetson and proposed developing a free Web-based e-mail system. The idea was to give a free e-mail account to anyone who wanted one, and to make the accounts accessible from the Web. In exchange for 15 percent of the company, Draper Fisher Jurvetson financed Bhatia and Smith, who formed a company called Hotmail. With three full-time people and 14 part-time people who worked for stock options, they were able to develop and launch the service in July 1996. Within a month after launch, they had 100,000 subscribers. In December 1997, less than 18 months after launching the service, Hotmail had over 12 million subscribers and was acquired by Microsoft, reportedly for \$400 million. The success of Hotmail is often attributed to its “first-mover advantage” and to the intrinsic “viral marketing” of e-mail. (Perhaps some of the students reading this book will be among the new entrepreneurs who conceive and develop first-mover Internet services with inherent viral marketing.)

Web e-mail continues to thrive, becoming more sophisticated and powerful every year. One of the most popular services today is Google’s gmail, which offers gigabytes of free storage, advanced spam filtering and virus detection, e-mail encryption (using SSL), mail fetching from third-party e-mail services, and a search-oriented interface. Asynchronous messaging within social networks, such as Facebook, has also become popular in recent years.

When Bob wants to access the messages in his mailbox, the mail server containing his mailbox authenticates Bob (with usernames and passwords). Alice's mail server must also deal with failures in Bob's mail server. If Alice's server cannot deliver mail to Bob's server, Alice's server holds the message in a **message queue** and attempts to transfer the message later. Reattempts are often done every 30 minutes or so; if there is no success after several days, the server removes the message and notifies the sender (Alice) with an e-mail message.

SMTP is the principal application-layer protocol for Internet electronic mail. It uses the reliable data transfer service of TCP to transfer mail from the sender's mail server to the recipient's mail server. As with most application-layer protocols, SMTP has two sides: a client side, which executes on the sender's mail server, and a server side, which executes on the recipient's mail server. Both the client and server sides of SMTP run on every mail server. When a mail server sends mail to other mail servers, it acts as an SMTP client. When a mail server receives mail from other mail servers, it acts as an SMTP server.

2.4.1 SMTP

SMTP, defined in RFC 5321, is at the heart of Internet electronic mail. As mentioned above, SMTP transfers messages from senders' mail servers to the recipients' mail servers. SMTP is much older than HTTP. (The original SMTP RFC dates back to 1982, and SMTP was around long before that.) Although SMTP has numerous wonderful qualities, as evidenced by its ubiquity in the Internet, it is nevertheless a legacy technology that possesses certain archaic characteristics. For example, it restricts the body (not just the headers) of all mail messages to simple 7-bit ASCII. This restriction made sense in the early 1980s when transmission capacity was scarce and no one was e-mailing large attachments or large image, audio, or video files. But today, in the multimedia era, the 7-bit ASCII restriction is a bit of a pain—it requires binary multimedia data to be encoded to ASCII before being sent over SMTP; and it requires the corresponding ASCII message to be decoded back to binary after SMTP transport. Recall from Section 2.2 that HTTP does not require multimedia data to be ASCII encoded before transfer.

To illustrate the basic operation of SMTP, let's walk through a common scenario. Suppose Alice wants to send Bob a simple ASCII message.

1. Alice invokes her user agent for e-mail, provides Bob's e-mail address (for example, `bob@someschool.edu`), composes a message, and instructs the user agent to send the message.
2. Alice's user agent sends the message to her mail server, where it is placed in a message queue.

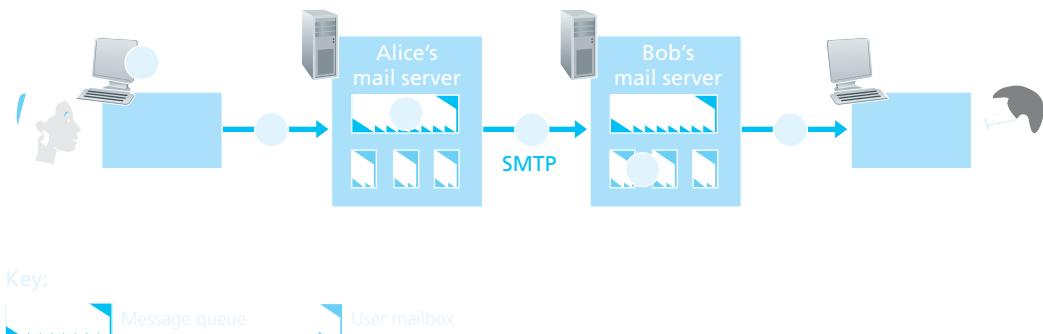


Figure 2.17 ♦ Alice sends a message to Bob

3. The client side of SMTP, running on Alice’s mail server, sees the message in the message queue. It opens a TCP connection to an SMTP server, running on Bob’s mail server.
4. After some initial SMTP handshaking, the SMTP client sends Alice’s message into the TCP connection.
5. At Bob’s mail server, the server side of SMTP receives the message. Bob’s mail server then places the message in Bob’s mailbox.
6. Bob invokes his user agent to read the message at his convenience.

The scenario is summarized in Figure 2.17.

It is important to observe that SMTP does not normally use intermediate mail servers for sending mail, even when the two mail servers are located at opposite ends of the world. If Alice’s server is in Hong Kong and Bob’s server is in St. Louis, the TCP connection is a direct connection between the Hong Kong and St. Louis servers. In particular, if Bob’s mail server is down, the message remains in Alice’s mail server and waits for a new attempt—the message does not get placed in some intermediate mail server.

Let’s now take a closer look at how SMTP transfers a message from a sending mail server to a receiving mail server. We will see that the SMTP protocol has many similarities with protocols that are used for face-to-face human interaction. First, the client SMTP (running on the sending mail server host) has TCP establish a connection to port 25 at the server SMTP (running on the receiving mail server host). If the server is down, the client tries again later. Once this connection is established, the server and client perform some application-layer handshaking—just as humans often introduce themselves before transferring information from one to another, SMTP clients and servers introduce themselves before transferring information. During this SMTP handshaking phase, the

SMTP client indicates the e-mail address of the sender (the person who generated the message) and the e-mail address of the recipient. Once the SMTP client and server have introduced themselves to each other, the client sends the message. SMTP can count on the reliable data transfer service of TCP to get the message to the server without errors. The client then repeats this process over the same TCP connection if it has other messages to send to the server; otherwise, it instructs TCP to close the connection.

Let's next take a look at an example transcript of messages exchanged between an SMTP client (C) and an SMTP server (S). The hostname of the client is `crepes.fr` and the hostname of the server is `hamburger.edu`. The ASCII text lines prefaced with C: are exactly the lines the client sends into its TCP socket, and the ASCII text lines prefaced with S: are exactly the lines the server sends into its TCP socket. The following transcript begins as soon as the TCP connection is established.

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr ... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

In the example above, the client sends a message (“Do you like ketchup? How about pickles?”) from mail server `crepes.fr` to mail server `hamburger.edu`. As part of the dialogue, the client issued five commands: HELO (an abbreviation for HELLO), MAIL FROM, RCPT TO, DATA, and QUIT. These commands are self-explanatory. The client also sends a line consisting of a single period, which indicates the end of the message to the server. (In ASCII jargon, each message ends with CRLF.CRLF, where CR and LF stand for carriage return and line feed, respectively.) The server issues replies to each command, with each reply having a reply code and some (optional) English-language explanation. We mention here that SMTP uses persistent connections: If the sending mail server has several messages to send to the same receiving mail server, it can send all of the messages over the same TCP connection. For each message, the client begins the process with

a new MAIL FROM: `crepes.fr`, designates the end of message with an isolated period, and issues QUIT only after all messages have been sent.

It is highly recommended that you use Telnet to carry out a direct dialogue with an SMTP server. To do this, issue

```
telnet serverName 25
```

where `serverName` is the name of a local mail server. When you do this, you are simply establishing a TCP connection between your local host and the mail server. After typing this line, you should immediately receive the 220 reply from the server. Then issue the SMTP commands HELO, MAIL FROM, RCPT TO, DATA, CRLF.CRLF, and QUIT at the appropriate times. It is also highly recommended that you do Programming Assignment 3 at the end of this chapter. In that assignment, you'll build a simple user agent that implements the client side of SMTP. It will allow you to send an e-mail message to an arbitrary recipient via a local mail server.

2.4.2 Comparison with HTTP

Let's now briefly compare SMTP with HTTP. Both protocols are used to transfer files from one host to another: HTTP transfers files (also called objects) from a Web server to a Web client (typically a browser); SMTP transfers files (that is, e-mail messages) from one mail server to another mail server. When transferring the files, both persistent HTTP and SMTP use persistent connections. Thus, the two protocols have common characteristics. However, there are important differences. First, HTTP is mainly a **pull protocol**—someone loads information on a Web server and users use HTTP to pull the information from the server at their convenience. In particular, the TCP connection is initiated by the machine that wants to receive the file. On the other hand, SMTP is primarily a **push protocol**—the sending mail server pushes the file to the receiving mail server. In particular, the TCP connection is initiated by the machine that wants to send the file.

A second difference, which we alluded to earlier, is that SMTP requires each message, including the body of each message, to be in 7-bit ASCII format. If the message contains characters that are not 7-bit ASCII (for example, French characters with accents) or contains binary data (such as an image file), then the message has to be encoded into 7-bit ASCII. HTTP data does not impose this restriction.

A third important difference concerns how a document consisting of text and images (along with possibly other media types) is handled. As we learned in Section 2.2, HTTP encapsulates each object in its own HTTP response message. Internet mail places all of the message's objects into one message.

2.4.3 Mail Message Formats

When Alice writes an ordinary snail-mail letter to Bob, she may include all kinds of peripheral header information at the top of the letter, such as Bob's address, her own return address, and the date. Similarly, when an e-mail message is sent from one person to another, a header containing peripheral information precedes the body of the message itself. This peripheral information is contained in a series of header lines, which are defined in RFC 5322. The header lines and the body of the message are separated by a blank line (that is, by CRLF). RFC 5322 specifies the exact format for mail header lines as well as their semantic interpretations. As with HTTP, each header line contains readable text, consisting of a keyword followed by a colon followed by a value. Some of the keywords are required and others are optional. Every header must have a `From:` header line and a `To:` header line; a header may include a `Subject:` header line as well as other optional header lines. It is important to note that these header lines are *different* from the SMTP commands we studied in Section 2.4.1 (even though they contain some common words such as “*from*” and “*to*”). The commands in that section were part of the SMTP handshaking protocol; the header lines examined in this section are part of the mail message itself.

A typical message header looks like this:

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Searching for the meaning of life.
```

After the message header, a blank line follows; then the message body (in ASCII) follows. You should use Telnet to send a message to a mail server that contains some header lines, including the `Subject:` header line. To do this, issue `telnet serverName 25`, as discussed in Section 2.4.1.

2.4.4 Mail Access Protocols

Once SMTP delivers the message from Alice's mail server to Bob's mail server, the message is placed in Bob's mailbox. Throughout this discussion we have tacitly assumed that Bob reads his mail by logging onto the server host and then executing a mail reader that runs on that host. Up until the early 1990s this was the standard way of doing things. But today, mail access uses a client-server architecture—the typical user reads e-mail with a client that executes on the user's end system, for example, on an office PC, a laptop, or a smartphone. By executing a mail client on a local PC, users enjoy a rich set of features, including the ability to view multimedia messages and attachments.

Given that Bob (the recipient) executes his user agent on his local PC, it is natural to consider placing a mail server on his local PC as well. With this approach,

Alice's mail server would dialogue directly with Bob's PC. There is a problem with this approach, however. Recall that a mail server manages mailboxes and runs the client and server sides of SMTP. If Bob's mail server were to reside on his local PC, then Bob's PC would have to remain always on, and connected to the Internet, in order to receive new mail, which can arrive at any time. This is impractical for many Internet users. Instead, a typical user runs a user agent on the local PC but accesses its mailbox stored on an always-on shared mail server. This mail server is shared with other users and is typically maintained by the user's ISP (for example, university or company).

Now let's consider the path an e-mail message takes when it is sent from Alice to Bob. We just learned that at some point along the path the e-mail message needs to be deposited in Bob's mail server. This could be done simply by having Alice's user agent send the message directly to Bob's mail server. And this could be done with SMTP—indeed, SMTP has been designed for pushing e-mail from one host to another. However, typically the sender's user agent does not dialogue directly with the recipient's mail server. Instead, as shown in Figure 2.18, Alice's user agent uses SMTP to push the e-mail message into her mail server, then Alice's mail server uses SMTP (as an SMTP client) to relay the e-mail message to Bob's mail server. Why the two-step procedure? Primarily because without relaying through Alice's mail server, Alice's user agent doesn't have any recourse to an unreachable destination mail server. By having Alice first deposit the e-mail in her own mail server, Alice's mail server can repeatedly try to send the message to Bob's mail server, say every 30 minutes, until Bob's mail server becomes operational. (And if Alice's mail server is down, then she has the recourse of complaining to her system administrator!) The SMTP RFC defines how the SMTP commands can be used to relay a message across multiple SMTP servers.

But there is still one missing piece to the puzzle! How does a recipient like Bob, running a user agent on his local PC, obtain his messages, which are sitting in a mail server within Bob's ISP? Note that Bob's user agent can't use SMTP to obtain the messages because obtaining the messages is a pull operation, whereas SMTP is a

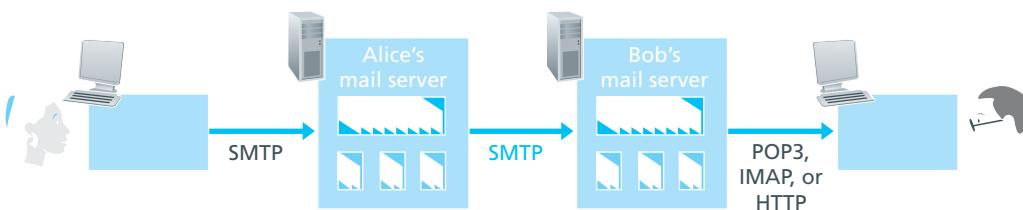


Figure 2.18 ♦ E-mail protocols and their communicating entities

push protocol. The puzzle is completed by introducing a special mail access protocol that transfers messages from Bob's mail server to his local PC. There are currently a number of popular mail access protocols, including **Post Office Protocol—Version 3 (POP3)**, **Internet Mail Access Protocol (IMAP)**, and **HTTP**.

Figure 2.18 provides a summary of the protocols that are used for Internet mail: SMTP is used to transfer mail from the sender's mail server to the recipient's mail server; SMTP is also used to transfer mail from the sender's user agent to the sender's mail server. A mail access protocol, such as POP3, is used to transfer mail from the recipient's mail server to the recipient's user agent.

POP3

POP3 is an extremely simple mail access protocol. It is defined in [RFC 1939], which is short and quite readable. Because the protocol is so simple, its functionality is rather limited. POP3 begins when the user agent (the client) opens a TCP connection to the mail server (the server) on port 110. With the TCP connection established, POP3 progresses through three phases: authorization, transaction, and update. During the first phase, authorization, the user agent sends a username and a password (in the clear) to authenticate the user. During the second phase, transaction, the user agent retrieves messages; also during this phase, the user agent can mark messages for deletion, remove deletion marks, and obtain mail statistics. The third phase, update, occurs after the client has issued the `quit` command, ending the POP3 session; at this time, the mail server deletes the messages that were marked for deletion.

In a POP3 transaction, the user agent issues commands, and the server responds to each command with a reply. There are two possible responses: `+OK` (sometimes followed by server-to-client data), used by the server to indicate that the previous command was fine; and `-ERR`, used by the server to indicate that something was wrong with the previous command.

The authorization phase has two principal commands: `user <username>` and `pass <password>`. To illustrate these two commands, we suggest that you Telnet directly into a POP3 server, using port 110, and issue these commands. Suppose that `mailServer` is the name of your mail server. You will see something like:

```
telnet mailServer 110
+OK POP3 server ready
user bob
+OK
pass hungry
+OK user successfully logged on
```

If you misspell a command, the POP3 server will reply with an `-ERR` message.

Now let's take a look at the transaction phase. A user agent using POP3 can often be configured (by the user) to “download and delete” or to “download and keep.” The sequence of commands issued by a POP3 user agent depends on which of these two modes the user agent is operating in. In the download-and-delete mode, the user agent will issue the `list`, `retr`, and `dele` commands. As an example, suppose the user has two messages in his or her mailbox. In the dialogue below, **C:** (standing for client) is the user agent and **S:** (standing for server) is the mail server. The transaction will look something like:

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: (blah blah ...
S: .....
S: .....blah)
S: .
C: dele 1
C: retr 2
S: (blah blah ...
S: .....
S: .....blah)
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

The user agent first asks the mail server to list the size of each of the stored messages. The user agent then retrieves and deletes each message from the server. Note that after the authorization phase, the user agent employed only four commands: `list`, `retr`, `dele`, and `quit`. The syntax for these commands is defined in RFC 1939. After processing the `quit` command, the POP3 server enters the update phase and removes messages 1 and 2 from the mailbox.

A problem with this download-and-delete mode is that the recipient, Bob, may be nomadic and may want to access his mail messages from multiple machines, for example, his office PC, his home PC, and his portable computer. The download-and-delete mode partitions Bob's mail messages over these three machines; in particular, if Bob first reads a message on his office PC, he will not be able to reread the message from his portable at home later in the evening. In the download-and-keep mode, the user agent leaves the messages on the mail server after downloading them. In this case, Bob can reread messages from different machines; he can access a message from work and access it again later in the week from home.

During a POP3 session between a user agent and the mail server, the POP3 server maintains some state information; in particular, it keeps track of which user messages have been marked deleted. However, the POP3 server does not carry state information across POP3 sessions. This lack of state information across sessions greatly simplifies the implementation of a POP3 server.

IMAP

With POP3 access, once Bob has downloaded his messages to the local machine, he can create mail folders and move the downloaded messages into the folders. Bob can then delete messages, move messages across folders, and search for messages (by sender name or subject). But this paradigm—namely, folders and messages in the local machine—poses a problem for the nomadic user, who would prefer to maintain a folder hierarchy on a remote server that can be accessed from any computer. This is not possible with POP3—the POP3 protocol does not provide any means for a user to create remote folders and assign messages to folders.

To solve this and other problems, the IMAP protocol, defined in [RFC 3501], was invented. Like POP3, IMAP is a mail access protocol. It has many more features than POP3, but it is also significantly more complex. (And thus the client and server side implementations are significantly more complex.)

An IMAP server will associate each message with a folder; when a message first arrives at the server, it is associated with the recipient's INBOX folder. The recipient can then move the message into a new, user-created folder, read the message, delete the message, and so on. The IMAP protocol provides commands to allow users to create folders and move messages from one folder to another. IMAP also provides commands that allow users to search remote folders for messages matching specific criteria. Note that, unlike POP3, an IMAP server maintains user state information across IMAP sessions—for example, the names of the folders and which messages are associated with which folders.

Another important feature of IMAP is that it has commands that permit a user agent to obtain components of messages. For example, a user agent can obtain just the message header of a message or just one part of a multipart MIME message. This feature is useful when there is a low-bandwidth connection (for example, a slow-speed modem link) between the user agent and its mail server. With a low-bandwidth connection, the user may not want to download all of the messages in its mailbox, particularly avoiding long messages that might contain, for example, an audio or video clip.

Web-Based E-Mail

More and more users today are sending and accessing their e-mail through their Web browsers. Hotmail introduced Web-based access in the mid 1990s. Now Web-based

e-mail is also provided by Google, Yahoo!, as well as just about every major university and corporation. With this service, the user agent is an ordinary Web browser, and the user communicates with its remote mailbox via HTTP. When a recipient, such as Bob, wants to access a message in his mailbox, the e-mail message is sent from Bob's mail server to Bob's browser using the HTTP protocol rather than the POP3 or IMAP protocol. When a sender, such as Alice, wants to send an e-mail message, the e-mail message is sent from her browser to her mail server over HTTP rather than over SMTP. Alice's mail server, however, still sends messages to, and receives messages from, other mail servers using SMTP.

2.5 DNS—The Internet's Directory Service

We human beings can be identified in many ways. For example, we can be identified by the names that appear on our birth certificates. We can be identified by our social security numbers. We can be identified by our driver's license numbers. Although each of these identifiers can be used to identify people, within a given context one identifier may be more appropriate than another. For example, the computers at the IRS (the infamous tax-collecting agency in the United States) prefer to use fixed-length social security numbers rather than birth certificate names. On the other hand, ordinary people prefer the more mnemonic birth certificate names rather than social security numbers. (Indeed, can you imagine saying, "Hi. My name is 132-67-9875. Please meet my husband, 178-87-1146.")

Just as humans can be identified in many ways, so too can Internet hosts. One identifier for a host is its **hostname**. Hostnames—such as `cnn.com`, `www.yahoo.com`, `gaia.cs.umass.edu`, and `cis.poly.edu`—are mnemonic and are therefore appreciated by humans. However, hostnames provide little, if any, information about the location within the Internet of the host. (A hostname such as `www.eurecom.fr`, which ends with the country code `.fr`, tells us that the host is probably in France, but doesn't say much more.) Furthermore, because hostnames can consist of variable-length alphanumeric characters, they would be difficult to process by routers. For these reasons, hosts are also identified by so-called **IP addresses**.

We discuss IP addresses in some detail in Chapter 4, but it is useful to say a few brief words about them now. An IP address consists of four bytes and has a rigid hierarchical structure. An IP address looks like `121.7.106.83`, where each period separates one of the bytes expressed in decimal notation from 0 to 255. An IP address is hierarchical because as we scan the address from left to right, we obtain more and more specific information about where the host is located in the Internet (that is, within which network, in the network of networks). Similarly, when we scan a postal address from bottom to top, we obtain more and more specific information about where the addressee is located.

2.5.1 Services Provided by DNS

We have just seen that there are two ways to identify a host—by a hostname and by an IP address. People prefer the more mnemonic hostname identifier, while routers prefer fixed-length, hierarchically structured IP addresses. In order to reconcile these preferences, we need a directory service that translates hostnames to IP addresses. This is the main task of the Internet's **domain name system (DNS)**. The DNS is (1) a distributed database implemented in a hierarchy of **DNS servers**, and (2) an application-layer protocol that allows hosts to query the distributed database. The DNS servers are often UNIX machines running the Berkeley Internet Name Domain (BIND) software [BIND 2012]. The DNS protocol runs over UDP and uses port 53.

DNS is commonly employed by other application-layer protocols—including HTTP, SMTP, and FTP—to translate user-supplied hostnames to IP addresses. As an example, consider what happens when a browser (that is, an HTTP client), running on some user's host, requests the URL `www.someschool.edu/index.html`. In order for the user's host to be able to send an HTTP request message to the Web server `www.someschool.edu`, the user's host must first obtain the IP address of `www.someschool.edu`. This is done as follows.

1. The same user machine runs the client side of the DNS application.
2. The browser extracts the hostname, `www.someschool.edu`, from the URL and passes the hostname to the client side of the DNS application.
3. The DNS client sends a query containing the hostname to a DNS server.
4. The DNS client eventually receives a reply, which includes the IP address for the hostname.
5. Once the browser receives the IP address from DNS, it can initiate a TCP connection to the HTTP server process located at port 80 at that IP address.

We see from this example that DNS adds an additional delay—sometimes substantial—to the Internet applications that use it. Fortunately, as we discuss below, the desired IP address is often cached in a “nearby” DNS server, which helps to reduce DNS network traffic as well as the average DNS delay.

DNS provides a few other important services in addition to translating hostnames to IP addresses:

- **Host aliasing.** A host with a complicated hostname can have one or more alias names. For example, a hostname such as `relay1.west-coast.enterprise.com` could have, say, two aliases such as `enterprise.com` and `www.enterprise.com`. In this case, the hostname `relay1.west-coast.enterprise.com` is said to be a **canonical hostname**. Alias hostnames, when present, are typically more mnemonic than canonical hostnames.



PRINCIPLES IN PRACTICE

DNS: CRITICAL NETWORK FUNCTIONS VIA THE CLIENT-SERVER PARADIGM

Like HTTP, FTP, and SMTP, the DNS protocol is an application-layer protocol since it (1) runs between communicating end systems using the client-server paradigm and (2) relies on an underlying end-to-end transport protocol to transfer DNS messages between communicating end systems. In another sense, however, the role of the DNS is quite different from Web, file transfer, and e-mail applications. Unlike these applications, the DNS is not an application with which a user directly interacts. Instead, the DNS provides a core Internet function—namely, translating hostnames to their underlying IP addresses, for user applications and other software in the Internet. We noted in Section 1.2 that much of the complexity in the Internet architecture is located at the “edges” of the network. The DNS, which implements the critical name-to-address translation process using clients and servers located at the edge of the network, is yet another example of that design philosophy.

DNS can be invoked by an application to obtain the canonical hostname for a supplied alias hostname as well as the IP address of the host.

- **Mail server aliasing.** For obvious reasons, it is highly desirable that e-mail addresses be mnemonic. For example, if Bob has an account with Hotmail, Bob’s e-mail address might be as simple as `bob@hotmail.com`. However, the hostname of the Hotmail mail server is more complicated and much less mnemonic than simply `hotmail.com` (for example, the canonical hostname might be something like `relay1.west-coast.hotmail.com`). DNS can be invoked by a mail application to obtain the canonical hostname for a supplied alias hostname as well as the IP address of the host. In fact, the MX record (see below) permits a company’s mail server and Web server to have identical (aliased) hostnames; for example, a company’s Web server and mail server can both be called `enterprise.com`.
- **Load distribution.** DNS is also used to perform load distribution among replicated servers, such as replicated Web servers. Busy sites, such as `cnn.com`, are replicated over multiple servers, with each server running on a different end system and each having a different IP address. For replicated Web servers, a *set* of IP addresses is thus associated with one canonical hostname. The DNS database contains this set of IP addresses. When clients make a DNS query for a name mapped to a set of addresses, the server responds with the entire set of IP addresses, but rotates the ordering of the addresses within each reply. Because a client typically sends its HTTP request message to the IP address that is listed first in the set, DNS rotation distributes the traffic among the replicated servers.

DNS rotation is also used for e-mail so that multiple mail servers can have the same alias name. Also, content distribution companies such as Akamai have used DNS in more sophisticated ways [Dilley 2002] to provide Web content distribution (see Chapter 7).

The DNS is specified in RFC 1034 and RFC 1035, and updated in several additional RFCs. It is a complex system, and we only touch upon key aspects of its operation here. The interested reader is referred to these RFCs and the book by Albitz and Liu [Albitz 1993]; see also the retrospective paper [Mockapetris 1988], which provides a nice description of the what and why of DNS, and [Mockapetris 2005].

2.5.2 Overview of How DNS Works

We now present a high-level overview of how DNS works. Our discussion will focus on the hostname-to-IP-address translation service.

Suppose that some application (such as a Web browser or a mail reader) running in a user's host needs to translate a hostname to an IP address. The application will invoke the client side of DNS, specifying the hostname that needs to be translated. (On many UNIX-based machines, `gethostbyname()` is the function call that an application calls in order to perform the translation.) DNS in the user's host then takes over, sending a query message into the network. All DNS query and reply messages are sent within UDP datagrams to port 53. After a delay, ranging from milliseconds to seconds, DNS in the user's host receives a DNS reply message that provides the desired mapping. This mapping is then passed to the invoking application. Thus, from the perspective of the invoking application in the user's host, DNS is a black box providing a simple, straightforward translation service. But in fact, the black box that implements the service is complex, consisting of a large number of DNS servers distributed around the globe, as well as an application-layer protocol that specifies how the DNS servers and querying hosts communicate.

A simple design for DNS would have one DNS server that contains all the mappings. In this centralized design, clients simply direct all queries to the single DNS server, and the DNS server responds directly to the querying clients. Although the simplicity of this design is attractive, it is inappropriate for today's Internet, with its vast (and growing) number of hosts. The problems with a centralized design include:

- **A single point of failure.** If the DNS server crashes, so does the entire Internet!
- **Traffic volume.** A single DNS server would have to handle all DNS queries (for all the HTTP requests and e-mail messages generated from hundreds of millions of hosts).

- **Distant centralized database.** A single DNS server cannot be “close to” all the querying clients. If we put the single DNS server in New York City, then all queries from Australia must travel to the other side of the globe, perhaps over slow and congested links. This can lead to significant delays.
 - **Maintenance.** The single DNS server would have to keep records for all Internet hosts. Not only would this centralized database be huge, but it would have to be updated frequently to account for every new host.

In summary, a centralized database in a single DNS server simply *doesn't scale*. Consequently, the DNS is distributed by design. In fact, the DNS is a wonderful example of how a distributed database can be implemented in the Internet.

A Distributed, Hierarchical Database

In order to deal with the issue of scale, the DNS uses a large number of servers, organized in a hierarchical fashion and distributed around the world. No single DNS server has all of the mappings for all of the hosts in the Internet. Instead, the mappings are distributed across the DNS servers. To a first approximation, there are three classes of DNS servers—root DNS servers, top-level domain (TLD) DNS servers, and authoritative DNS servers—organized in a hierarchy as shown in Figure 2.19. To understand how these three classes of servers interact, suppose a DNS client wants to determine the IP address for the hostname `www.amazon.com`. To a first approximation, the following events will take place. The client first contacts one of the root servers, which returns IP addresses for TLD servers for the top-level domain `.com`. The client then contacts one of these TLD servers, which returns the IP address of an authoritative server for `amazon.com`. Finally, the client contacts one of the authoritative servers for `amazon.com`, which returns the IP address

Figure 2.19 ♦ Portion of the hierarchy of DNS servers

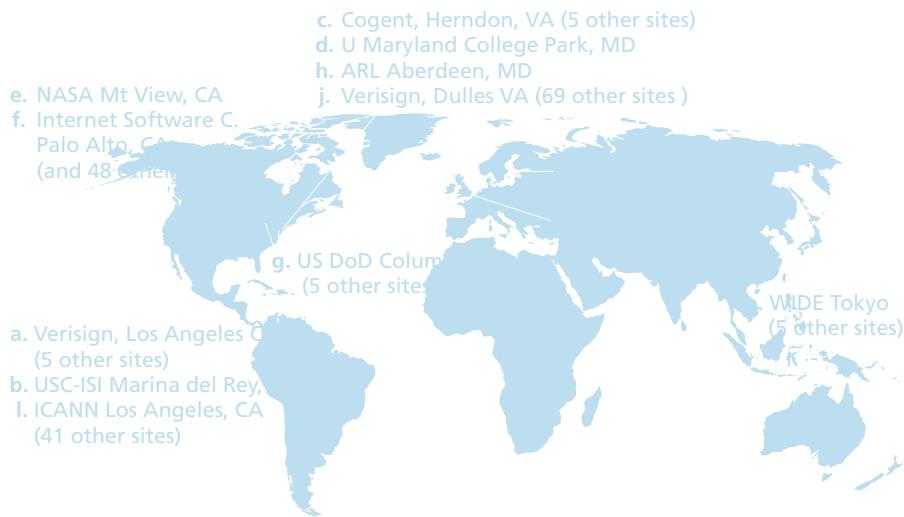


Figure 2.20 ♦ DNS root servers in 2012 (name, organization, location)

for the hostname `www.amazon.com`. We'll soon examine this DNS lookup process in more detail. But let's first take a closer look at these three classes of DNS servers:

- **Root DNS servers.** In the Internet there are 13 root DNS servers (labeled A through M), most of which are located in North America. An October 2006 map of the root DNS servers is shown in Figure 2.20; a list of the current root DNS servers is available via [Root-servers 2012]. Although we have referred to each of the 13 root DNS servers as if it were a single server, each “server” is actually a network of replicated servers, for both security and reliability purposes. All together, there are 247 root servers as of fall 2011.
- **Top-level domain (TLD) servers.** These servers are responsible for top-level domains such as com, org, net, edu, and gov, and all of the country top-level domains such as uk, fr, ca, and jp. The company Verisign Global Registry Services maintains the TLD servers for the com top-level domain, and the company Educause maintains the TLD servers for the edu top-level domain. See [IANA TLD 2012] for a list of all top-level domains.
- **Authoritative DNS servers.** Every organization with publicly accessible hosts (such as Web servers and mail servers) on the Internet must provide publicly accessible DNS records that map the names of those hosts to IP addresses. An organization's authoritative DNS server houses these DNS records. An organization can

choose to implement its own authoritative DNS server to hold these records; alternatively, the organization can pay to have these records stored in an authoritative DNS server of some service provider. Most universities and large companies implement and maintain their own primary and secondary (backup) authoritative DNS server.

The root, TLD, and authoritative DNS servers all belong to the hierarchy of DNS servers, as shown in Figure 2.19. There is another important type of DNS server called the **local DNS server**. A local DNS server does not strictly belong to the hierarchy of servers but is nevertheless central to the DNS architecture. Each ISP—such as a university, an academic department, an employee’s company, or a residential ISP—has a local DNS server (also called a default name server). When a host connects to an ISP, the ISP provides the host with the IP addresses of one or more of its local DNS servers (typically through DHCP, which is discussed in Chapter 4). You can easily determine the IP address of your local DNS server by accessing network status windows in Windows or UNIX. A host’s local DNS server is typically “close to” the host. For an institutional ISP, the local DNS server may be on the same LAN as the host; for a residential ISP, it is typically separated from the host by no more than a few routers. When a host makes a DNS query, the query is sent to the local DNS server, which acts a proxy, forwarding the query into the DNS server hierarchy, as we’ll discuss in more detail below.

Let’s take a look at a simple example. Suppose the host `cis.poly.edu` desires the IP address of `gaia.cs.umass.edu`. Also suppose that Polytechnic’s local DNS server is called `dns.poly.edu` and that an authoritative DNS server for `gaia.cs.umass.edu` is called `dns.umass.edu`. As shown in Figure 2.21, the host `cis.poly.edu` first sends a DNS query message to its local DNS server, `dns.poly.edu`. The query message contains the hostname to be translated, namely, `gaia.cs.umass.edu`. The local DNS server forwards the query message to a root DNS server. The root DNS server takes note of the `edu` suffix and returns to the local DNS server a list of IP addresses for TLD servers responsible for `edu`. The local DNS server then resends the query message to one of these TLD servers. The TLD server takes note of the `umass.edu` suffix and responds with the IP address of the authoritative DNS server for the University of Massachusetts, namely, `dns.umass.edu`. Finally, the local DNS server resends the query message directly to `dns.umass.edu`, which responds with the IP address of `gaia.cs.umass.edu`. Note that in this example, in order to obtain the mapping for one hostname, eight DNS messages were sent: four query messages and four reply messages! We’ll soon see how DNS caching reduces this query traffic.

Our previous example assumed that the TLD server knows the authoritative DNS server for the hostname. In general this is not always true. Instead, the TLD server may know only of an intermediate DNS server, which in turn knows the authoritative DNS server for the hostname. For example, suppose again that the University of

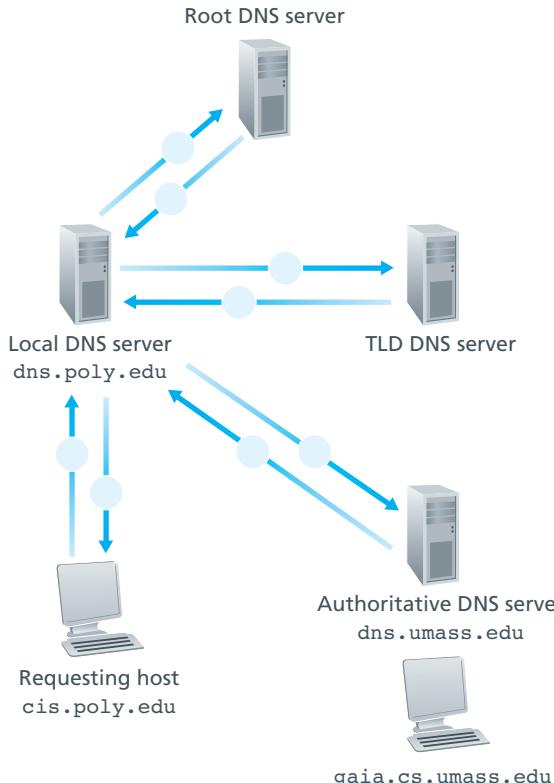


Figure 2.21 ♦ Interaction of the various DNS servers

Massachusetts has a DNS server for the university, called `dns.umass.edu`. Also suppose that each of the departments at the University of Massachusetts has its own DNS server, and that each departmental DNS server is authoritative for all hosts in the department. In this case, when the intermediate DNS server, `dns.umass.edu`, receives a query for a host with a hostname ending with `cs.umass.edu`, it returns to `dns.poly.edu` the IP address of `dns.cs.umass.edu`, which is authoritative for all hostnames ending with `cs.umass.edu`. The local DNS server `dns.poly.edu` then sends the query to the authoritative DNS server, which returns the desired mapping to the local DNS server, which in turn returns the mapping to the requesting host. In this case, a total of 10 DNS messages are sent!

The example shown in Figure 2.21 makes use of both **recursive queries** and **iterative queries**. The query sent from `cis.poly.edu` to `dns.poly.edu` is a recursive query, since the query asks `dns.poly.edu` to obtain the mapping on its

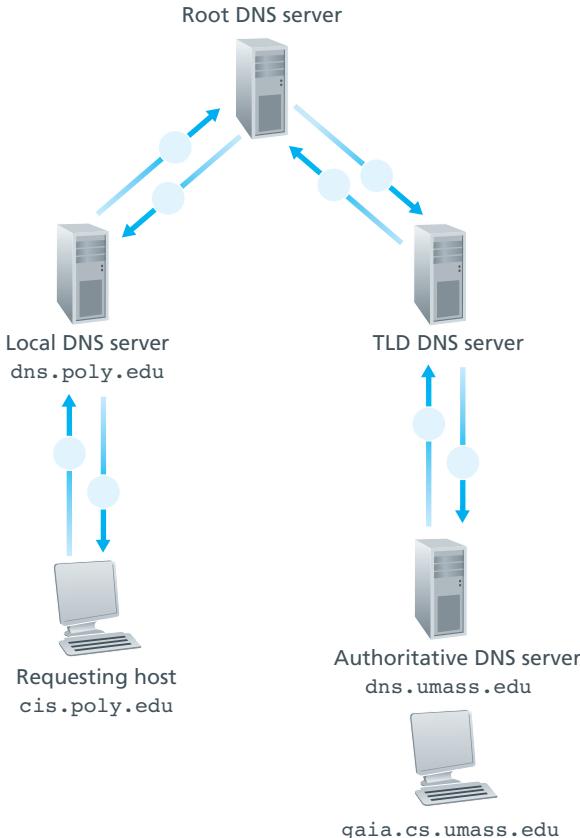


Figure 2.22 ♦ Recursive queries in DNS

behalf. But the subsequent three queries are iterative since all of the replies are directly returned to `dns.poly.edu`. In theory, any DNS query can be iterative or recursive. For example, Figure 2.22 shows a DNS query chain for which all of the queries are recursive. In practice, the queries typically follow the pattern in Figure 2.21: The query from the requesting host to the local DNS server is recursive, and the remaining queries are iterative.

DNS Caching

Our discussion thus far has ignored **DNS caching**, a critically important feature of the DNS system. In truth, DNS extensively exploits DNS caching in order to improve the delay performance and to reduce the number of DNS messages ricocheting around

the Internet. The idea behind DNS caching is very simple. In a query chain, when a DNS server receives a DNS reply (containing, for example, a mapping from a hostname to an IP address), it can cache the mapping in its local memory. For example, in Figure 2.21, each time the local DNS server `dns.poly.edu` receives a reply from some DNS server, it can cache any of the information contained in the reply. If a hostname/IP address pair is cached in a DNS server and another query arrives to the DNS server for the same hostname, the DNS server can provide the desired IP address, even if it is not authoritative for the hostname. Because hosts and mappings between hostnames and IP addresses are by no means permanent, DNS servers discard cached information after a period of time (often set to two days).

As an example, suppose that a host `apricot.poly.edu` queries `dns.poly.edu` for the IP address for the hostname `cnn.com`. Furthermore, suppose that a few hours later, another Polytechnic University host, say, `kiwi.poly.fr`, also queries `dns.poly.edu` with the same hostname. Because of caching, the local DNS server will be able to immediately return the IP address of `cnn.com` to this second requesting host without having to query any other DNS servers. A local DNS server can also cache the IP addresses of TLD servers, thereby allowing the local DNS server to bypass the root DNS servers in a query chain (this often happens).

2.5.3 DNS Records and Messages

The DNS servers that together implement the DNS distributed database store **resource records (RRs)**, including RRs that provide hostname-to-IP address mappings. Each DNS reply message carries one or more resource records. In this and the following subsection, we provide a brief overview of DNS resource records and messages; more details can be found in [Abitz 1993] or in the DNS RFCs [RFC 1034; RFC 1035].

A resource record is a four-tuple that contains the following fields:

(`Name`, `Value`, `Type`, `TTL`)

`TTL` is the time to live of the resource record; it determines when a resource should be removed from a cache. In the example records given below, we ignore the `TTL` field. The meaning of `Name` and `Value` depend on `Type`:

- If `Type=A`, then `Name` is a hostname and `Value` is the IP address for the hostname. Thus, a Type A record provides the standard hostname-to-IP address mapping. As an example, (`relay1.bar.foo.com`, `145.37.93.126`, `A`) is a Type A record.
- If `Type=NS`, then `Name` is a domain (such as `foo.com`) and `Value` is the hostname of an authoritative DNS server that knows how to obtain the IP addresses for hosts in the domain. This record is used to route DNS queries further along in

the query chain. As an example, (`foo.com`, `dns.foo.com`, `NS`) is a Type NS record.

- If `Type=CNAME`, then `Value` is a canonical hostname for the alias hostname `Name`. This record can provide querying hosts the canonical name for a hostname. As an example, (`foo.com`, `relay1.bar.foo.com`, `CNAME`) is a CNAME record.
- If `Type=MX`, then `Value` is the canonical name of a mail server that has an alias hostname `Name`. As an example, (`foo.com`, `mail.bar.foo.com`, `MX`) is an MX record. MX records allow the hostnames of mail servers to have simple aliases. Note that by using the MX record, a company can have the same aliased name for its mail server and for one of its other servers (such as its Web server). To obtain the canonical name for the mail server, a DNS client would query for an MX record; to obtain the canonical name for the other server, the DNS client would query for the CNAME record.

If a DNS server is authoritative for a particular hostname, then the DNS server will contain a Type A record for the hostname. (Even if the DNS server is not authoritative, it may contain a Type A record in its cache.) If a server is not authoritative for a hostname, then the server will contain a Type NS record for the domain that includes the hostname; it will also contain a Type A record that provides the IP address of the DNS server in the `Value` field of the NS record. As an example, suppose an edu TLD server is not authoritative for the host `gaia.cs.umass.edu`. Then this server will contain a record for a domain that includes the host `gaia.cs.umass.edu`, for example, (`umass.edu`, `dns.umass.edu`, `NS`). The edu TLD server would also contain a Type A record, which maps the DNS server `dns.umass.edu` to an IP address, for example, (`dns.umass.edu`, `128.119.40.111`, `A`).

DNS Messages

Earlier in this section, we referred to DNS query and reply messages. These are the only two kinds of DNS messages. Furthermore, both query and reply messages have the same format, as shown in Figure 2.23. The semantics of the various fields in a DNS message are as follows:

- The first 12 bytes is the *header section*, which has a number of fields. The first field is a 16-bit number that identifies the query. This identifier is copied into the reply message to a query, allowing the client to match received replies with sent queries. There are a number of flags in the flag field. A 1-bit query/reply flag indicates whether the message is a query (0) or a reply (1). A 1-bit authoritative flag is set in a reply message when a DNS server is an authoritative server for a queried name. A 1-bit recursion-desired flag is set when a client (host or DNS server) desires that the DNS server perform recursion when it doesn't have the record. A 1-bit recursion-available field is set in a reply if the DNS server supports recursion. In the header,

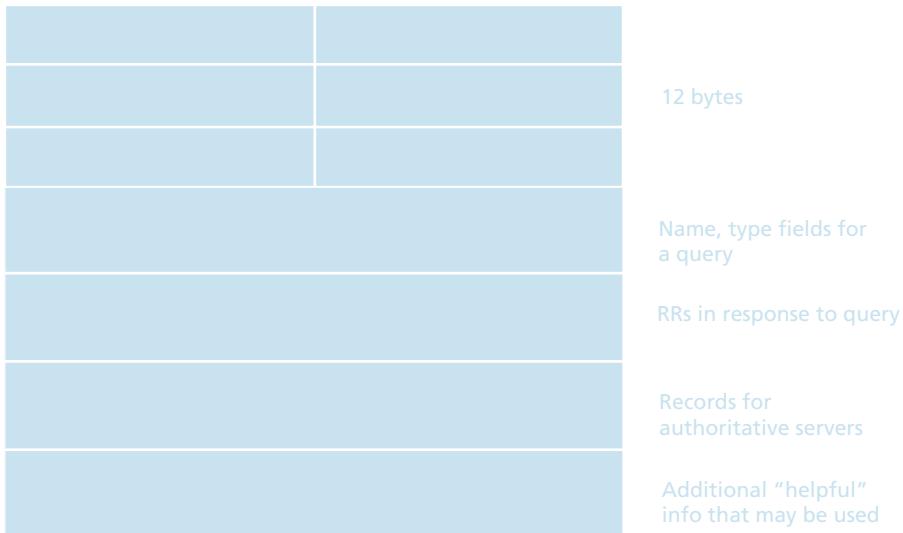


Figure 2.23 ♦ DNS message format

there are also four number-of-fields. These fields indicate the number of occurrences of the four types of data sections that follow the header.

- The *question section* contains information about the query that is being made. This section includes (1) a name field that contains the name that is being queried, and (2) a type field that indicates the type of question being asked about the name—for example, a host address associated with a name (Type A) or the mail server for a name (Type MX).
- In a reply from a DNS server, the *answer section* contains the resource records for the name that was originally queried. Recall that in each resource record there is the **Type** (for example, A, NS, CNAME, and MX), the **Value**, and the **TTL**. A reply can return multiple RRs in the answer, since a hostname can have multiple IP addresses (for example, for replicated Web servers, as discussed earlier in this section).
- The *authority section* contains records of other authoritative servers.
- The *additional section* contains other helpful records. For example, the answer field in a reply to an MX query contains a resource record providing the canonical hostname of a mail server. The additional section contains a Type A record providing the IP address for the canonical hostname of the mail server.

How would you like to send a DNS query message directly from the host you're working on to some DNS server? This can easily be done with the **nslookup**

program, which is available from most Windows and UNIX platforms. For example, from a Windows host, open the Command Prompt and invoke the nslookup program by simply typing “nslookup.” After invoking nslookup, you can send a DNS query to any DNS server (root, TLD, or authoritative). After receiving the reply message from the DNS server, nslookup will display the records included in the reply (in a human-readable format). As an alternative to running nslookup from your own host, you can visit one of many Web sites that allow you to remotely employ nslookup. (Just type “nslookup” into a search engine and you’ll be brought to one of these sites.) The DNS Wireshark lab at the end of this chapter will allow you to explore the DNS in much more detail.

Inserting Records into the DNS Database

The discussion above focused on how records are retrieved from the DNS database. You might be wondering how records get into the database in the first place. Let’s look at how this is done in the context of a specific example. Suppose you have just created an exciting new startup company called Network Utopia. The first thing you’ll surely want to do is register the domain name `networkutopia.com` at a registrar. A **registrar** is a commercial entity that verifies the uniqueness of the domain name, enters the domain name into the DNS database (as discussed below), and collects a small fee from you for its services. Prior to 1999, a single registrar, Network Solutions, had a monopoly on domain name registration for `.com`, `.net`, and `.org` domains. But now there are many registrars competing for customers, and the Internet Corporation for Assigned Names and Numbers (ICANN) accredits the various registrars. A complete list of accredited registrars is available at <http://www.internic.net>.

When you register the domain name `networkutopia.com` with some registrar, you also need to provide the registrar with the names and IP addresses of your primary and secondary authoritative DNS servers. Suppose the names and IP addresses are `dns1.networkutopia.com`, `dns2.networkutopia.com`, `212.212.212.1`, and `212.212.212.2`. For each of these two authoritative DNS servers, the registrar would then make sure that a Type NS and a Type A record are entered into the TLD com servers. Specifically, for the primary authoritative server for `networkutopia.com`, the registrar would insert the following two resource records into the DNS system:

(`networkutopia.com, dns1.networkutopia.com, NS`)

(`dns1.networkutopia.com, 212.212.212.1, A`)

You’ll also have to make sure that the Type A resource record for your Web server `www.networkutopia.com` and the Type MX resource record for your mail server `mail.networkutopia.com` are entered into your authoritative DNS servers. (Until recently, the contents of each DNS server were configured statically,



FOCUS ON SECURITY

DNS VULNERABILITIES

We have seen that DNS is a critical component of the Internet infrastructure, with many important services - including the Web and e-mail - simply incapable of functioning without it. We therefore naturally ask, how can DNS be attacked? Is DNS a sitting duck, waiting to be knocked out of service, while taking most Internet applications down with it?

The first type of attack that comes to mind is a DDoS bandwidth-flooding attack (see Section 1.6) against DNS servers. For example, an attacker could attempt to send to each DNS root server a deluge of packets, so many that the majority of legitimate DNS queries never get answered. Such a large-scale DDoS attack against DNS root servers actually took place on October 21, 2002. In this attack, the attackers leveraged a botnet to send truck loads of ICMP ping messages to each of the 13 DNS root servers. (ICMP messages are discussed in Chapter 4. For now, it suffices to know that ICMP packets are special types of IP datagrams.) Fortunately, this large-scale attack caused minimal damage, having little or no impact on users' Internet experience. The attackers did succeed at directing a deluge of packets at the root servers. But many of the DNS root servers were protected by packet filters, configured to always block all ICMP ping messages directed at the root servers. These protected servers were thus spared and functioned as normal. Furthermore, most local DNS servers cache the IP addresses of top-level-domain servers, allowing the query process to often bypass the DNS root servers.

A potentially more effective DDoS attack against DNS would be to send a deluge of DNS queries to top-level-domain servers, for example, to all the top-level-domain servers that handle the .com domain. It would be harder to filter DNS queries directed to DNS servers; and top-level-domain servers are not as easily bypassed as are root servers. But the severity of such an attack would be partially mitigated by caching in local DNS servers.

DNS could potentially be attacked in other ways. In a man-in-the-middle attack, the attacker intercepts queries from hosts and returns bogus replies. In the DNS poisoning attack, the attacker sends bogus replies to a DNS server, tricking the server into accepting bogus records into its cache. Either of these attacks could be used, for example, to redirect an unsuspecting Web user to the attacker's Web site. These attacks, however, are difficult to implement, as they require intercepting packets or throttling servers [Skoudis 2006].

Another important DNS attack is not an attack on the DNS service per se, but instead exploits the DNS infrastructure to launch a DDoS attack against a targeted host (for example, your university's mail server). In this attack, the attacker sends DNS queries to many authoritative DNS servers, with each query having the spoofed source address of the targeted host. The DNS servers then send their replies directly to the targeted host. If the queries can be crafted in such a way that a response is much larger



FOCUS ON SECURITY

(in bytes) than a query (so-called amplification), then the attacker can potentially overwhelm the target without having to generate much of its own traffic. Such reflection attacks exploiting DNS have had limited success to date [Mirkovic 2005].

In summary, DNS has demonstrated itself to be surprisingly robust against attacks. To date, there hasn't been an attack that has successfully impeded the DNS service. There have been successful reflector attacks; however, these attacks can be (and are being) addressed by appropriate configuration of DNS servers.

for example, from a configuration file created by a system manager. More recently, an UPDATE option has been added to the DNS protocol to allow data to be dynamically added or deleted from the database via DNS messages. [RFC 2136] and [RFC 3007] specify DNS dynamic updates.)

Once all of these steps are completed, people will be able to visit your Web site and send e-mail to the employees at your company. Let's conclude our discussion of DNS by verifying that this statement is true. This verification also helps to solidify what we have learned about DNS. Suppose Alice in Australia wants to view the Web page www.networkkutopia.com. As discussed earlier, her host will first send a DNS query to her local DNS server. The local DNS server will then contact a TLD com server. (The local DNS server will also have to contact a root DNS server if the address of a TLD com server is not cached.) This TLD server contains the Type NS and Type A resource records listed above, because the registrar had these resource records inserted into all of the TLD com servers. The TLD com server sends a reply to Alice's local DNS server, with the reply containing the two resource records. The local DNS server then sends a DNS query to 212.212.212.1, asking for the Type A record corresponding to www.networkkutopia.com. This record provides the IP address of the desired Web server, say, 212.212.71.4, which the local DNS server passes back to Alice's host. Alice's browser can now initiate a TCP connection to the host 212.212.71.4 and send an HTTP request over the connection. Whew! There's a lot more going on than what meets the eye when one surfs the Web!

2.6 Peer-to-Peer Applications

The applications described in this chapter thus far—including the Web, e-mail, and DNS—all employ client-server architectures with significant reliance on always-on infrastructure servers. Recall from Section 2.1.1 that with a P2P architecture, there is minimal (or no) reliance on always-on infrastructure servers. Instead, pairs of intermittently connected hosts, called peers, communicate directly with each other.

The peers are not owned by a service provider, but are instead desktops and laptops controlled by users.

In this section we'll examine two different applications that are particularly well-suited for P2P designs. The first is file distribution, where the application distributes a file from a single source to a large number of peers. File distribution is a nice place to start our investigation of P2P, as it clearly exposes the self-scalability of P2P architectures. As a specific example for file distribution, we'll describe the popular BitTorrent system. The second P2P application we'll examine is a database distributed over a large community of peers. For this application, we'll explore the concept of a Distributed Hash Table (DHT).

2.6.1 P2P File Distribution

We begin our foray into P2P by considering a very natural application, namely, distributing a large file from a single server to a large number of hosts (called peers). The file might be a new version of the Linux operating system, a software patch for an existing operating system or application, an MP3 music file, or an MPEG video file. In client-server file distribution, the server must send a copy of the file to each of the peers—placing an enormous burden on the server and consuming a large amount of server bandwidth. In P2P file distribution, each peer can redistribute any portion of the file it has received to any other peers, thereby assisting the server in the distribution process. As of 2012, the most popular P2P file distribution protocol is BitTorrent. Originally developed by Bram Cohen, there are now many different independent BitTorrent clients conforming to the BitTorrent protocol, just as there are a number of Web browser clients that conform to the HTTP protocol. In this subsection, we first examine the self-scalability of P2P architectures in the context of file distribution. We then describe BitTorrent in some detail, highlighting its most important characteristics and features.

Scalability of P2P Architectures

To compare client-server architectures with peer-to-peer architectures, and illustrate the inherent self-scalability of P2P, we now consider a simple quantitative model for distributing a file to a fixed set of peers for both architecture types. As shown in Figure 2.24, the server and the peers are connected to the Internet with access links. Denote the upload rate of the server's access link by u_s , the upload rate of the i th peer's access link by u_i , and the download rate of the i th peer's access link by d_i . Also denote the size of the file to be distributed (in bits) by F and the number of peers that want to obtain a copy of the file by N . The **distribution time** is the time it takes to get a copy of the file to all N peers. In our analysis of the distribution time below, for both client-server and P2P architectures, we make the simplifying (and generally accurate [Akella 2003]) assumption that the Internet core has abundant

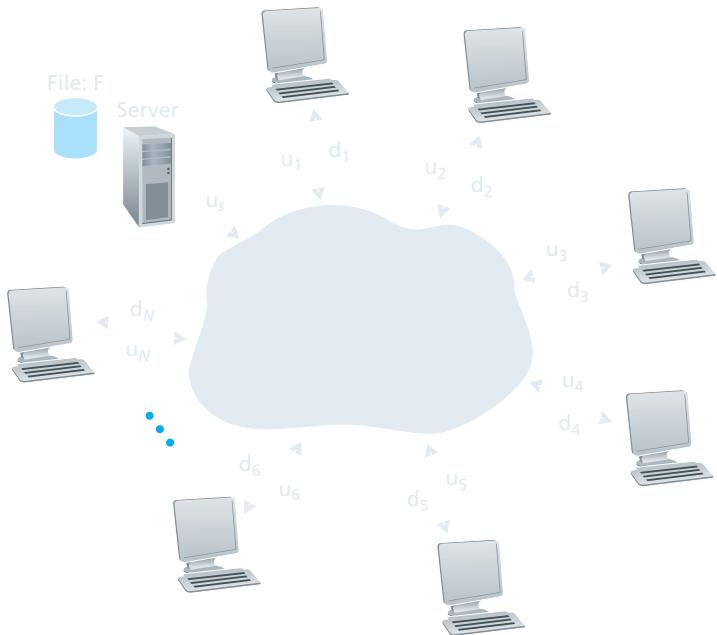


Figure 2.24 ♦ An illustrative file distribution problem

bandwidth, implying that all of the bottlenecks are in access networks. We also suppose that the server and clients are not participating in any other network applications, so that all of their upload and download access bandwidth can be fully devoted to distributing this file.

Let's first determine the distribution time for the client-server architecture, which we denote by D_{cs} . In the client-server architecture, none of the peers aids in distributing the file. We make the following observations:

- The server must transmit one copy of the file to each of the N peers. Thus the server must transmit NF bits. Since the server's upload rate is u_s , the time to distribute the file must be at least NF/u_s .
- Let d_{\min} denote the download rate of the peer with the lowest download rate, that is, $d_{\min} = \min\{d_1, d_p, \dots, d_N\}$. The peer with the lowest download rate cannot obtain all F bits of the file in less than F/d_{\min} seconds. Thus the minimum distribution time is at least F/d_{\min} .

Putting these two observations together, we obtain

$$D_{cs} \geq \max \left\{ \frac{NF}{u_s}, \frac{F}{d_{\min}} \right\}.$$

This provides a lower bound on the minimum distribution time for the client-server architecture. In the homework problems you will be asked to show that the server can schedule its transmissions so that the lower bound is actually achieved. So let's take this lower bound provided above as the actual distribution time, that is,

$$D_{cs} = \max \left\{ \frac{NF}{u_s}, \frac{F}{d_{min}} \right\} \quad (2.1)$$

We see from Equation 2.1 that for N large enough, the client-server distribution time is given by NF/u_s . Thus, the distribution time increases linearly with the number of peers N . So, for example, if the number of peers from one week to the next increases a thousand-fold from a thousand to a million, the time required to distribute the file to all peers increases by 1,000.

Let's now go through a similar analysis for the P2P architecture, where each peer can assist the server in distributing the file. In particular, when a peer receives some file data, it can use its own upload capacity to redistribute the data to other peers. Calculating the distribution time for the P2P architecture is somewhat more complicated than for the client-server architecture, since the distribution time depends on how each peer distributes portions of the file to the other peers. Nevertheless, a simple expression for the minimal distribution time can be obtained [Kumar 2006]. To this end, we first make the following observations:

- At the beginning of the distribution, only the server has the file. To get this file into the community of peers, the server must send each bit of the file at least once into its access link. Thus, the minimum distribution time is at least F/u_s . (Unlike the client-server scheme, a bit sent once by the server may not have to be sent by the server again, as the peers may redistribute the bit among themselves.)
- As with the client-server architecture, the peer with the lowest download rate cannot obtain all F bits of the file in less than F/d_{min} seconds. Thus the minimum distribution time is at least F/d_{min} .
- Finally, observe that the total upload capacity of the system as a whole is equal to the upload rate of the server plus the upload rates of each of the individual peers, that is, $u_{total} = u_s + u_1 + \dots + u_N$. The system must deliver (upload) F bits to each of the N peers, thus delivering a total of NF bits. This cannot be done at a rate faster than u_{total} . Thus, the minimum distribution time is also at least $NF/(u_s + u_1 + \dots + u_N)$.

Putting these three observations together, we obtain the minimum distribution time for P2P, denoted by D_{P2P} .

$$D_{P2P} \geq \max \left\{ \frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{u_s + \sum_{i=1}^N u_i} \right\} \quad (2.2)$$

Equation 2.2 provides a lower bound for the minimum distribution time for the P2P architecture. It turns out that if we imagine that each peer can redistribute a bit as soon as it receives the bit, then there is a redistribution scheme that actually achieves this lower bound [Kumar 2006]. (We will prove a special case of this result in the homework.) In reality, where chunks of the file are redistributed rather than individual bits, Equation 2.2 serves as a good approximation of the actual minimum distribution time. Thus, let's take the lower bound provided by Equation 2.2 as the actual minimum distribution time, that is,

$$D_{\text{P2P}} = \max \left\{ \frac{F}{u_s}, \frac{F}{d_{\min}}, \frac{NF}{u_s + \sum_{i=1}^N u_i} \right\} \quad (2.3)$$

Figure 2.25 compares the minimum distribution time for the client-server and P2P architectures assuming that all peers have the same upload rate u . In Figure 2.25, we have set $F/u = 1$ hour, $u_s = 10u$, and $d_{\min} \geq u_s$. Thus, a peer can transmit the entire file in one hour, the server transmission rate is 10 times the peer upload rate, and (for simplicity) the peer download rates are set large enough so as not to have an effect. We see from Figure 2.25 that for the client-server architecture, the distribution time increases linearly and without bound as the number of peers increases. However, for the P2P architecture, the minimal distribution time is not only always less than the distribution time of the client-server architecture; it is also less than one hour for *any* number of peers N . Thus, applications with the P2P architecture can be self-scaling. This scalability is a direct consequence of peers being redistributors as well as consumers of bits.



Figure 2.25 ♦ Distribution time for P2P and client-server architectures

BitTorrent

BitTorrent is a popular P2P protocol for file distribution [Chao 2011]. In BitTorrent lingo, the collection of all peers participating in the distribution of a particular file is called a *torrent*. Peers in a torrent download equal-size *chunks* of the file from one another, with a typical chunk size of 256 KBytes. When a peer first joins a torrent, it has no chunks. Over time it accumulates more and more chunks. While it downloads chunks it also uploads chunks to other peers. Once a peer has acquired the entire file, it may (selfishly) leave the torrent, or (altruistically) remain in the torrent and continue to upload chunks to other peers. Also, any peer may leave the torrent at any time with only a subset of chunks, and later rejoin the torrent.

Let's now take a closer look at how BitTorrent operates. Since BitTorrent is a rather complicated protocol and system, we'll only describe its most important mechanisms, sweeping some of the details under the rug; this will allow us to see the forest through the trees. Each torrent has an infrastructure node called a *tracker*. When a peer joins a torrent, it registers itself with the tracker and periodically informs the tracker that it is still in the torrent. In this manner, the tracker keeps track of the peers that are participating in the torrent. A given torrent may have fewer than ten or more than a thousand peers participating at any instant of time.

As shown in Figure 2.26, when a new peer, Alice, joins the torrent, the tracker randomly selects a subset of peers (for concreteness, say 50) from the set of participating peers, and sends the IP addresses of these 50 peers to Alice. Possessing this list of peers, Alice attempts to establish concurrent TCP connections with all the peers on this list. Let's call all the peers with which Alice succeeds in establishing a TCP connection "neighboring peers." (In Figure 2.26, Alice is shown to have only three neighboring peers. Normally, she would have many more.) As time evolves, some of these peers may leave and other peers (outside the initial 50) may attempt to establish TCP connections with Alice. So a peer's neighboring peers will fluctuate over time.

At any given time, each peer will have a subset of chunks from the file, with different peers having different subsets. Periodically, Alice will ask each of her neighboring peers (over the TCP connections) for the list of the chunks they have. If Alice has L different neighbors, she will obtain L lists of chunks. With this knowledge, Alice will issue requests (again over the TCP connections) for chunks she currently does not have.

So at any given instant of time, Alice will have a subset of chunks and will know which chunks her neighbors have. With this information, Alice will have two important decisions to make. First, which chunks should she request first from her neighbors? And second, to which of her neighbors should she send requested chunks? In deciding which chunks to request, Alice uses a technique called **rarest first**. The idea is to determine, from among the chunks she does not have, the chunks that are the rarest among her neighbors (that is, the chunks that have the fewest repeated copies among her neighbors) and then request those rarest chunks first. In this manner, the rarest chunks get more quickly redistributed, aiming to (roughly) equalize the numbers of copies of each chunk in the torrent.

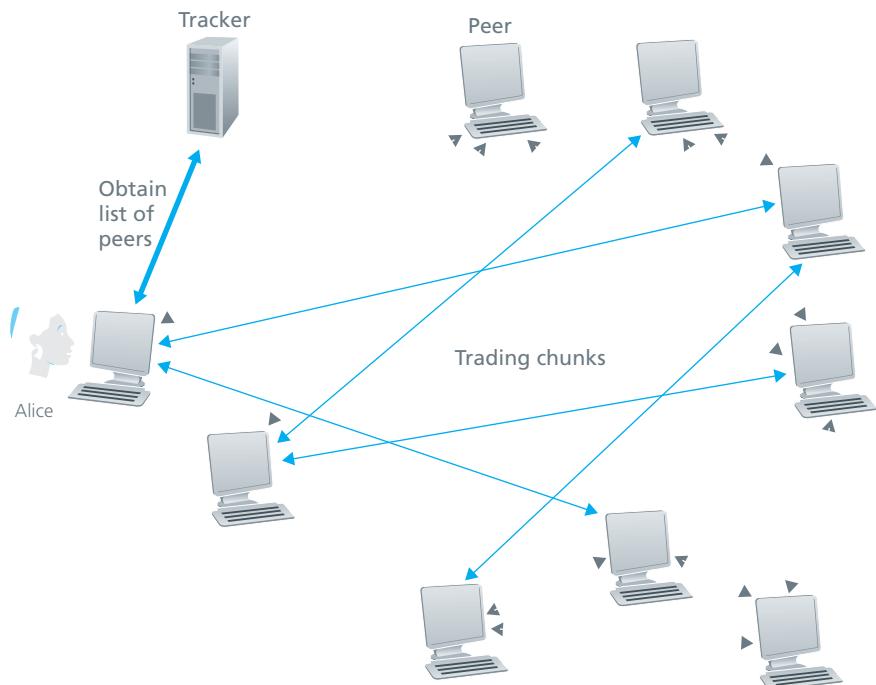


Figure 2.26 ♦ File distribution with BitTorrent

To determine which requests she responds to, BitTorrent uses a clever trading algorithm. The basic idea is that Alice gives priority to the neighbors that are currently supplying her data *at the highest rate*. Specifically, for each of her neighbors, Alice continually measures the rate at which she receives bits and determines the four peers that are feeding her bits at the highest rate. She then reciprocates by sending chunks to these same four peers. Every 10 seconds, she recalculates the rates and possibly modifies the set of four peers. In BitTorrent lingo, these four peers are said to be **unchoked**. Importantly, every 30 seconds, she also picks one additional neighbor at random and sends it chunks. Let's call the randomly chosen peer Bob. In BitTorrent lingo, Bob is said to be **optimistically unchoked**. Because Alice is sending data to Bob, she may become one of Bob's top four uploaders, in which case Bob would start to send data to Alice. If the rate at which Bob sends data to Alice is high enough, Bob could then, in turn, become one of Alice's top four uploaders. In other words, every 30 seconds, Alice will randomly choose a new trading partner and initiate trading with that partner. If the two peers are satisfied with the trading, they will put each other in their top four lists and continue trading with each other until one of the peers finds a better partner. The effect is that peers capable of uploading at compatible rates tend to find each other. The random neighbor selection also allows new peers to get

chunks, so that they can have something to trade. All other neighboring peers besides these five peers (four “top” peers and one probing peer) are “choked,” that is, they do not receive any chunks from Alice. BitTorrent has a number of interesting mechanisms that are not discussed here, including pieces (mini-chunks), pipelining, random first selection, endgame mode, and anti-snubbing [Cohen 2003].

The incentive mechanism for trading just described is often referred to as tit-for-tat [Cohen 2003]. It has been shown that this incentive scheme can be circumvented [Liogkas 2006; Locher 2006; Piatek 2007]. Nevertheless, the BitTorrent ecosystem is wildly successful, with millions of simultaneous peers actively sharing files in hundreds of thousands of torrents. If BitTorrent had been designed without tit-for-tat (or a variant), but otherwise exactly the same, BitTorrent would likely not even exist now, as the majority of the users would have been freeriders [Saroiu 2002].

Interesting variants of the BitTorrent protocol are proposed [Guo 2005; Piatek 2007]. Also, many of the P2P live streaming applications, such as PPLive and ppstream, have been inspired by BitTorrent [Hei 2007].

2.6.2 Distributed Hash Tables (DHTs)

In this section, we will consider how to implement a simple database in a P2P network. Let’s begin by describing a centralized version of this simple database, which will simply contain (key, value) pairs. For example, the keys could be social security numbers and the values could be the corresponding human names; in this case, an example key-value pair is (156-45-7081, Johnny Wu). Or the keys could be content names (e.g., names of movies, albums, and software), and the value could be the IP address at which the content is stored; in this case, an example key-value pair is (Led Zeppelin IV, 128.17.123.38). We query the database with a key. If there are one or more key-value pairs in the database that match the query key, the database returns the corresponding values. So, for example, if the database stores social security numbers and their corresponding human names, we can query with a specific social security number, and the database returns the name of the human who has that social security number. Or, if the database stores content names and their corresponding IP addresses, we can query with a specific content name, and the database returns the IP addresses that store the specific content.

Building such a database is straightforward with a client-server architecture that stores all the (key, value) pairs in one central server. So in this section, we’ll instead consider how to build a distributed, P2P version of this database that will store the (key, value) pairs over millions of peers. In the P2P system, each peer will only hold a small subset of the totality of the (key, value) pairs. We’ll allow any peer to query the distributed database with a particular key. The distributed database will then locate the peers that have the corresponding (key, value) pairs and return the key-value pairs to the querying peer. Any peer will also be allowed to insert new key-value pairs into the database. Such a distributed database is referred to as a **distributed hash table (DHT)**.



Before describing how we can create a DHT, let's first describe a specific example DHT service in the context of P2P file sharing. In this case, a key is the content name and the value is the IP address of a peer that has a copy of the content. So, if Bob and Charlie each have a copy of the latest Linux distribution, then the DHT database will include the following two key-value pairs: $(\text{Linux}, \text{IP}_{\text{Bob}})$ and $(\text{Linux}, \text{IP}_{\text{Charlie}})$. More specifically, since the DHT database is distributed over the peers, some peer, say Dave, will be responsible for the key “Linux” and will have the corresponding key-value pairs. Now suppose Alice wants to obtain a copy of Linux. Clearly, she first needs to know which peers have a copy of Linux before she can begin to download it. To this end, she queries the DHT with “Linux” as the key. The DHT then determines that the peer Dave is responsible for the key “Linux.” The DHT then contacts peer Dave, obtains from Dave the key-value pairs $(\text{Linux}, \text{IP}_{\text{Bob}})$ and $(\text{Linux}, \text{IP}_{\text{Charlie}})$, and passes them on to Alice. Alice can then download the latest Linux distribution from either IP_{Bob} or $\text{IP}_{\text{Charlie}}$.

Now let's return to the general problem of designing a DHT for general key-value pairs. One naïve approach to building a DHT is to randomly scatter the (key, value) pairs across all the peers and have each peer maintain a list of the IP addresses of all participating peers. In this design, the querying peer sends its query to all other peers, and the peers containing the (key, value) pairs that match the key can respond with their matching pairs. Such an approach is completely unscalable, of course, as it would require each peer to not only know about all other peers (possibly millions of such peers!) but even worse, have each query sent to *all* peers.

We now describe an elegant approach to designing a DHT. To this end, let's first assign an identifier to each peer, where each identifier is an integer in the range $[0, 2^n - 1]$ for some fixed n . Note that each such identifier can be expressed by an n -bit representation. Let's also require each key to be an integer in the same range. The astute reader may have observed that the example keys described a little earlier (social security numbers and content names) are not integers. To create integers out of such keys, we will use a hash function that maps each key (e.g., social security number) to an integer in the range $[0, 2^n - 1]$. A hash function is a many-to-one function for which two different inputs can have the same output (same integer), but the likelihood of the having the same output is extremely small. (Readers who are unfamiliar with hash functions may want to visit Chapter 7, in which hash functions are discussed in some detail.) The hash function is assumed to be available to all peers in the system. Henceforth, when we refer to the “key,” we are referring to the hash of the original key. So, for example, if the original key is “Led Zeppelin IV,” the key used in the DHT will be the integer that equals the hash of “Led Zeppelin IV.” As you may have guessed, this is why “Hash” is used in the term “Distributed Hash Function.”

Let's now consider the problem of storing the (key, value) pairs in the DHT. The central issue here is defining a rule for assigning keys to peers. Given that each peer has an integer identifier and that each key is also an integer in the same range, a natural approach is to assign each (key, value) pair to the peer whose identifier is the *closest* to the key. To implement such a scheme, we'll need to define what is meant by “closest,” for which many conventions are possible. For convenience, let's define the

closest peer as the *closest successor of the key*. To gain some insight here, let's take a look at a specific example. Suppose $n = 4$ so that all the peer and key identifiers are in the range $[0, 15]$. Further suppose that there are eight peers in the system with identifiers 1, 3, 4, 5, 8, 10, 12, and 15. Finally, suppose we want to store the (key, value) pair (11, Johnny Wu) in one of the eight peers. But in which peer? Using our closest convention, since peer 12 is the closest successor for key 11, we therefore store the pair (11, Johnny Wu) in the peer 12. [To complete our definition of closest, if the key is exactly equal to one of the peer identifiers, we store the (key, value) pair in that matching peer; and if the key is larger than all the peer identifiers, we use a modulo- 2^n convention, storing the (key, value) pair in the peer with the smallest identifier.]

Now suppose a peer, Alice, wants to insert a (key, value) pair into the DHT. Conceptually, this is straightforward: She first determines the peer whose identifier is closest to the key; she then sends a message to that peer, instructing it to store the (key, value) pair. But how does Alice determine the peer that is closest to the key? If Alice were to keep track of all the peers in the system (peer IDs and corresponding IP addresses), she could locally determine the closest peer. But such an approach requires *each* peer to keep track of *all* other peers in the DHT—which is completely impractical for a large-scale system with millions of peers.

Circular DHT

To address this problem of scale, let's now consider organizing the peers into a circle. In this circular arrangement, each peer only keeps track of its immediate successor and immediate predecessor (modulo 2^n). An example of such a circle is shown in Figure 2.27(a). In this example, n is again 4 and there are the same eight

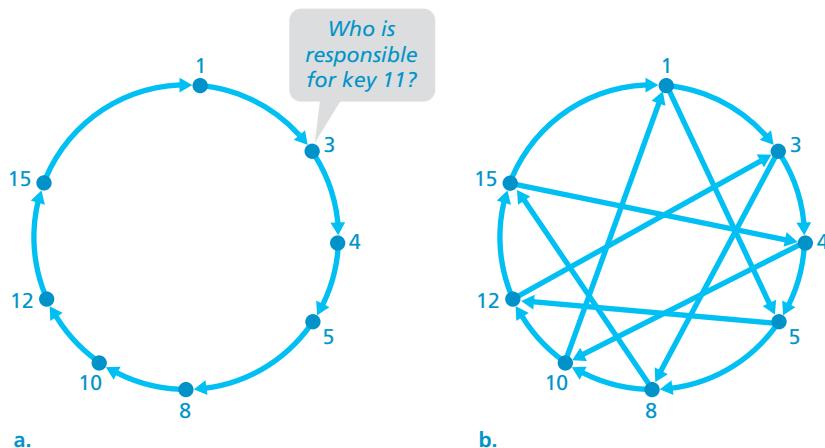


Figure 2.27 ♦ (a) A circular DHT. Peer 3 wants to determine who is responsible for key 11. (b) A circular DHT with shortcuts

peers from the previous example. Each peer is only aware of its immediate successor and predecessor; for example, peer 5 knows the IP address and identifier for peers 8 and 4 but does not necessarily know anything about any other peers that may be in the DHT. This circular arrangement of the peers is a special case of an **overlay network**. In an overlay network, the peers form an abstract logical network which resides above the “underlay” computer network consisting of physical links, routers, and hosts. The links in an overlay network are not physical links, but are simply virtual liaisons between pairs of peers. In the overlay in Figure 2.27(a), there are eight peers and eight overlay links; in the overlay in Figure 2.27(b) there are eight peers and 16 overlay links. A single overlay link typically uses many physical links and physical routers in the underlay network.

Using the circular overlay in Figure 2.27(a), now suppose that peer 3 wants to determine which peer in the DHT is responsible for key 11. Using the circular overlay, the origin peer (peer 3) creates a message saying “Who is responsible for key 11?” and sends this message clockwise around the circle. Whenever a peer receives such a message, because it knows the identifier of its successor and predecessor, it can determine whether it is responsible for (that is, closest to) the key in question. If a peer is not responsible for the key, it simply sends the message to its successor. So, for example, when peer 4 receives the message asking about key 11, it determines that it is not responsible for the key (because its successor is closer to the key), so it just passes the message along to peer 5. This process continues until the message arrives at peer 12, who determines that it is the closest peer to key 11. At this point, peer 12 can send a message back to the querying peer, peer 3, indicating that it is responsible for key 11.

The circular DHT provides a very elegant solution for reducing the amount of overlay information each peer must manage. In particular, each peer needs only to be aware of two peers, its immediate successor and its immediate predecessor. But this solution introduces yet a new problem. Although each peer is only aware of two neighboring peers, to find the node responsible for a key (in the worst case), all N nodes in the DHT will have to forward a message around the circle; $N/2$ messages are sent on average.

Thus, in designing a DHT, there is tradeoff between the number of neighbors each peer has to track and the number of messages that the DHT needs to send to resolve a single query. On one hand, if each peer tracks all other peers (mesh overlay), then only one message is sent per query, but each peer has to keep track of N peers. On the other hand, with a circular DHT, each peer is only aware of two peers, but $N/2$ messages are sent on average for each query. Fortunately, we can refine our designs of DHTs so that the number of neighbors per peer as well as the number of messages per query is kept to an acceptable size. One such refinement is to use the circular overlay as a foundation, but add “shortcuts” so that each peer not only keeps track of its immediate successor and predecessor, but also of a relatively small number of shortcut peers scattered about the circle. An example of such a circular DHT with some shortcuts is shown in Figure 2.27(b). Shortcuts are used to expedite the routing of query messages. Specifically, when a peer receives a message that is querying for a key, it forwards the

message to the neighbor (successor neighbor or one of the shortcut neighbors) which is the closest to the key. Thus, in Figure 2.27(b), when peer 4 receives the message asking about key 11, it determines that the closest peer to the key (among its neighbors) is its shortcut neighbor 10 and then forwards the message directly to peer 10. Clearly, shortcuts can significantly reduce the number of messages used to process a query.

The next natural question is “How many shortcut neighbors should a peer have, and which peers should be these shortcut neighbors? This question has received significant attention in the research community [Balakrishnan 2003; Androulidakis-Theotokis 2004]. Importantly, it has been shown that the DHT can be designed so that both the number of neighbors per peer as well as the number of messages per query is $O(\log N)$, where N is the number of peers. Such designs strike a satisfactory compromise between the extreme solutions of using mesh and circular overlay topologies.

Peer Churn

In P2P systems, a peer can come or go without warning. Thus, when designing a DHT, we also must be concerned about maintaining the DHT overlay in the presence of such peer churn. To get a big-picture understanding of how this could be accomplished, let’s once again consider the circular DHT in Figure 2.27(a). To handle peer churn, we will now require each peer to track (that is, know the IP address of) its first and second successors; for example, peer 4 now tracks both peer 5 and peer 8. We also require each peer to periodically verify that its two successors are alive (for example, by periodically sending ping messages to them and asking for responses). Let’s now consider how the DHT is maintained when a peer abruptly leaves. For example, suppose peer 5 in Figure 2.27(a) abruptly leaves. In this case, the two peers preceding the departed peer (4 and 3) learn that 5 has departed, since it no longer responds to ping messages. Peers 4 and 3 thus need to update their successor state information. Let’s consider how peer 4 updates its state:

1. Peer 4 replaces its first successor (peer 5) with its second successor (peer 8).
2. Peer 4 then asks its new first successor (peer 8) for the identifier and IP address of its immediate successor (peer 10). Peer 4 then makes peer 10 its second successor.

In the homework problems, you will be asked to determine how peer 3 updates its overlay routing information.

Having briefly addressed what has to be done when a peer leaves, let’s now consider what happens when a peer wants to join the DHT. Let’s say a peer with identifier 13 wants to join the DHT, and at the time of joining, it only knows about peer 1’s existence in the DHT. Peer 13 would first send peer 1 a message, saying “what will be 13’s predecessor and successor?” This message gets forwarded through the DHT until it reaches peer 12, who realizes that it will be 13’s predecessor and that its current successor, peer 15, will become 13’s successor. Next, peer 12 sends this predecessor and successor information to peer 13. Peer 13 can now join

the DHT by making peer 15 its successor and by notifying peer 12 that it should change its immediate successor to 13.

DHTs have been finding widespread use in practice. For example, BitTorrent uses the Kademlia DHT to create a distributed tracker. In the BitTorrent, the key is the torrent identifier and the value is the IP addresses of all the peers currently participating in the torrent [Falkner 2007, Neglia 2007]. In this manner, by querying the DHT with a torrent identifier, a newly arriving BitTorrent peer can determine the peer that is responsible for the identifier (that is, for tracking the peers in the torrent). After having found that peer, the arriving peer can query it for a list of other peers in the torrent.

2.7 Socket Programming: Creating Network Applications

Now that we've looked at a number of important network applications, let's explore how network application programs are actually created. Recall from Section 2.1 that a typical network application consists of a pair of programs—a client program and a server program—residing in two different end systems. When these two programs are executed, a client process and a server process are created, and these processes communicate with each other by reading from, and writing to, sockets. When creating a network application, the developer's main task is therefore to write the code for both the client and server programs.

There are two types of network applications. One type is an implementation whose operation is specified in a protocol standard, such as an RFC or some other standards document; such an application is sometimes referred to as “open,” since the rules specifying its operation are known to all. For such an implementation, the client and server programs must conform to the rules dictated by the RFC. For example, the client program could be an implementation of the client side of the FTP protocol, described in Section 2.3 and explicitly defined in RFC 959; similarly, the server program could be an implementation of the FTP server protocol, also explicitly defined in RFC 959. If one developer writes code for the client program and another developer writes code for the server program, and both developers carefully follow the rules of the RFC, then the two programs will be able to interoperate. Indeed, many of today's network applications involve communication between client and server programs that have been created by independent developers—for example, a Firefox browser communicating with an Apache Web server, or a BitTorrent client communicating with BitTorrent tracker.

The other type of network application is a proprietary network application. In this case the client and server programs employ an application-layer protocol that has *not* been openly published in an RFC or elsewhere. A single developer (or

development team) creates both the client and server programs, and the developer has complete control over what goes in the code. But because the code does not implement an open protocol, other independent developers will not be able to develop code that interoperates with the application.

In this section, we'll examine the key issues in developing a client-server application, and we'll “get our hands dirty” by looking at code that implements a very simple client-server application. During the development phase, one of the first decisions the developer must make is whether the application is to run over TCP or over UDP. Recall that TCP is connection oriented and provides a reliable byte-stream channel through which data flows between two end systems. UDP is connectionless and sends independent packets of data from one end system to the other, without any guarantees about delivery. Recall also that when a client or server program implements a protocol defined by an RFC, it should use the well-known port number associated with the protocol; conversely, when developing a proprietary application, the developer must be careful to avoid using such well-known port numbers. (Port numbers were briefly discussed in Section 2.1. They are covered in more detail in Chapter 3.)

We introduce UDP and TCP socket programming by way of a simple UDP application and a simple TCP application. We present the simple UDP and TCP applications in Python. We could have written the code in Java, C, or C++, but we chose Python mostly because Python clearly exposes the key socket concepts. With Python there are fewer lines of code, and each line can be explained to the novice programmer without difficulty. But there's no need to be frightened if you are not familiar with Python. You should be able to easily follow the code if you have experience programming in Java, C, or C++.

If you are interested in client-server programming with Java, you are encouraged to see the companion Web site for this textbook; in fact, you can find there all the examples in this section (and associated labs) in Java. For readers who are interested in client-server programming in C, there are several good references available [Donahoo 2001; Stevens 1997; Frost 1994; Kurose 1996]; our Python examples below have a similar look and feel to C.

2.7.1 Socket Programming with UDP

In this subsection, we'll write simple client-server programs that use UDP; in the following section, we'll write similar programs that use TCP.

Recall from Section 2.1 that processes running on different machines communicate with each other by sending messages into sockets. We said that each process is analogous to a house and the process's socket is analogous to a door. The application resides on one side of the door in the house; the transport-layer protocol resides on the other side of the door in the outside world. The application developer has control of everything on the application-layer side of the socket; however, it has little control of the transport-layer side.

Now let's take a closer look at the interaction between two communicating processes that use UDP sockets. Before the sending process can push a packet of data out the socket door, when using UDP, it must first attach a destination address to the packet. After the packet passes through the sender's socket, the Internet will use this destination address to route the packet through the Internet to the socket in the receiving process. When the packet arrives at the receiving socket, the receiving process will retrieve the packet through the socket, and then inspect the packet's contents and take appropriate action.

So you may be now wondering, what goes into the destination address that is attached to the packet? As you might expect, the destination host's IP address is part of the destination address. By including the destination IP address in the packet, the routers in the Internet will be able to route the packet through the Internet to the destination host. But because a host may be running many network application processes, each with one or more sockets, it is also necessary to identify the particular socket in the destination host. When a socket is created, an identifier, called a **port number**, is assigned to it. So, as you might expect, the packet's destination address also includes the socket's port number. In summary, the sending process attaches to the packet a destination address which consists of the destination host's IP address and the destination socket's port number. Moreover, as we shall soon see, the sender's source address—consisting of the IP address of the source host and the port number of the source socket—are also attached to the packet. However, attaching the source address to the packet is typically *not* done by the UDP application code; instead it is automatically done by the underlying operating system.

We'll use the following simple client-server application to demonstrate socket programming for both UDP and TCP:

1. The client reads a line of characters (data) from its keyboard and sends the data to the server.
2. The server receives the data and converts the characters to uppercase.
3. The server sends the modified data to the client.
4. The client receives the modified data and displays the line on its screen.

Figure 2.28 highlights the main socket-related activity of the client and server that communicate over the UDP transport service.

Now let's get our hands dirty and take a look at the client-server program pair for a UDP implementation of this simple application. We also provide a detailed, line-by-line analysis after each program. We'll begin with the UDP client, which will send a simple application-level message to the server. In order for the server to be able to receive and reply to the client's message, it must be ready and running—that is, it must be running as a process before the client sends its message.

The client program is called `UDPClient.py`, and the server program is called `UDPServer.py`. In order to emphasize the key issues, we intentionally provide code that is minimal. “Good code” would certainly have a few more auxiliary lines, in

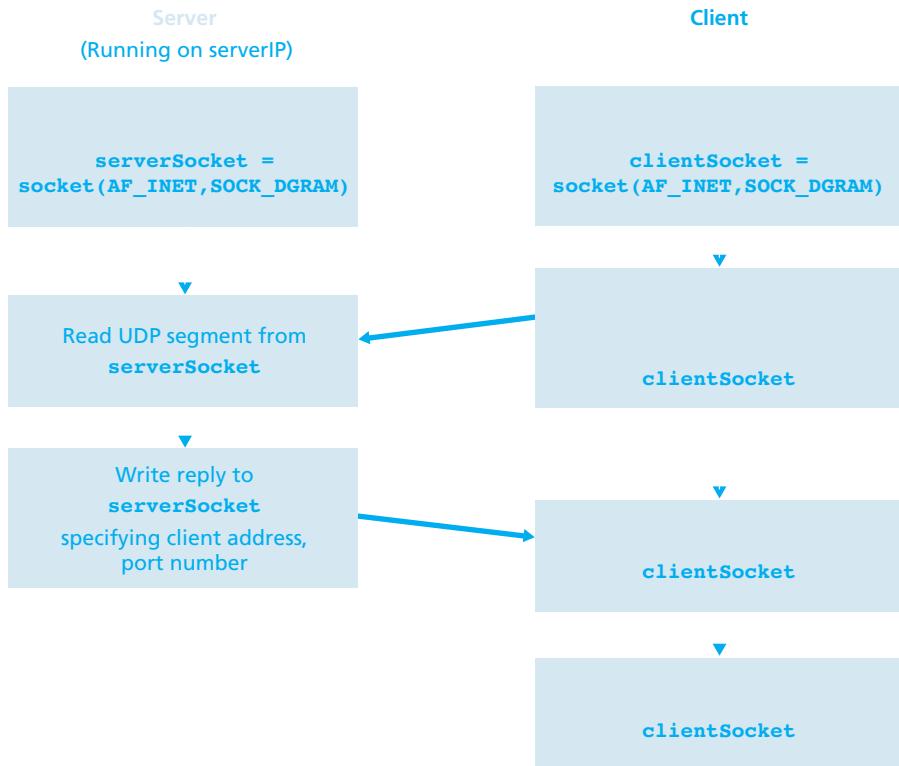


Figure 2.28 ♦ The client-server application using UDP

particular for handling error cases. For this application, we have arbitrarily chosen 12000 for the server port number.

UDPClient.py

Here is the code for the client side of the application:

```
from socket import *
serverName = 'hostname'
serverPort = 12000
clientSocket = socket(socket.AF_INET, socket.SOCK_DGRAM)
message = raw_input('Input lowercase sentence:')
clientSocket.sendto(message,(serverName, serverPort))
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
print modifiedMessage
clientSocket.close()
```

Now let's take a look at the various lines of code in `UDPClient.py`.

```
from socket import *
```

The `socket` module forms the basis of all network communications in Python. By including this line, we will be able to create sockets within our program.

```
serverName = 'hostname'  
serverPort = 12000
```

The first line sets the string `serverName` to `hostname`. Here, we provide a string containing either the IP address of the server (e.g., “128.138.32.126”) or the hostname of the server (e.g., “`cis.poly.edu`”). If we use the hostname, then a DNS lookup will automatically be performed to get the IP address.) The second line sets the integer variable `serverPort` to 12000.

```
clientSocket = socket(socket.AF_INET, socket.SOCK_DGRAM)
```

This line creates the client's socket, called `clientSocket`. The first parameter indicates the address family; in particular, `AF_INET` indicates that the underlying network is using IPv4. (Do not worry about this now—we will discuss IPv4 in Chapter 4.) The second parameter indicates that the socket is of type `SOCK_DGRAM`, which means it is a UDP socket (rather than a TCP socket). Note that we are not specifying the port number of the client socket when we create it; we are instead letting the operating system do this for us. Now that the client process's door has been created, we will want to create a message to send through the door.

```
message = raw_input('Input lowercase sentence:')
```

`raw_input()` is a built-in function in Python. When this command is executed, the user at the client is prompted with the words “Input data:” The user then uses her keyboard to input a line, which is put into the variable `message`. Now that we have a socket and a message, we will want to send the message through the socket to the destination host.

```
clientSocket.sendto(message, (serverName, serverPort))
```

In the above line, the method `sendto()` attaches the destination address (`serverName, serverPort`) to the message and sends the resulting packet into the process's socket, `clientSocket`. (As mentioned earlier, the source address is also attached to the packet, although this is done automatically rather than explicitly by the code.) Sending a client-to-server message via a UDP socket is that simple! After sending the packet, the client waits to receive data from the server.

```
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
```

With the above line, when a packet arrives from the Internet at the client's socket, the packet's data is put into the variable `modifiedMessage` and the packet's source address is put into the variable `serverAddress`. The variable `serverAddress` contains both the server's IP address and the server's port number. The program `UDPClient` doesn't actually need this server address information, since it already knows the server address from the outset; but this line of Python provides the server address nevertheless. The method `recvfrom` also takes the buffer size 2048 as input. (This buffer size works for most purposes.)

```
print modifiedMessage
```

This line prints out `modifiedMessage` on the user's display. It should be the original line that the user typed, but now capitalized.

```
clientSocket.close()
```

This line closes the socket. The process then terminates.

UDPServer.py

Let's now take a look at the server side of the application:

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print "The server is ready to receive"
while 1:
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.upper()
    serverSocket.sendto(modifiedMessage, clientAddress)
```

Note that the beginning of `UDPServer` is similar to `UDPClient`. It also imports the `socket` module, also sets the integer variable `serverPort` to 12000, and also creates a socket of type `SOCK_DGRAM` (a UDP socket). The first line of code that is significantly different from `UDPClient` is:

```
serverSocket.bind(('', serverPort))
```

The above line binds (that is, assigns) the port number 12000 to the server's socket. Thus in `UDPServer`, the code (written by the application developer) is explicitly

assigning a port number to the socket. In this manner, when anyone sends a packet to port 12000 at the IP address of the server, that packet will be directed to this socket. UDPServer then enters a while loop; the while loop will allow UDPServer to receive and process packets from clients indefinitely. In the while loop, UDPServer waits for a packet to arrive.

```
message, clientAddress = serverSocket.recvfrom(2048)
```

This line of code is similar to what we saw in UDPClient. When a packet arrives at the server's socket, the packet's data is put into the variable `message` and the packet's source address is put into the variable `clientAddress`. The variable `clientAddress` contains both the client's IP address and the client's port number. Here, UDPServer *will* make use of this address information, as it provides a return address, similar to the return address with ordinary postal mail. With this source address information, the server now knows to where it should direct its reply.

```
modifiedMessage = message.upper()
```

This line is the heart of our simple application. It takes the line sent by the client and uses the method `upper()` to capitalize it.

```
serverSocket.sendto(modifiedMessage, clientAddress)
```

This last line attaches the client's address (IP address and port number) to the capitalized message, and sends the resulting packet into the server's socket. (As mentioned earlier, the server address is also attached to the packet, although this is done automatically rather than explicitly by the code.) The Internet will then deliver the packet to this client address. After the server sends the packet, it remains in the while loop, waiting for another UDP packet to arrive (from any client running on any host).

To test the pair of programs, you install and compile UDPClient.py in one host and UDPServer.py in another host. Be sure to include the proper hostname or IP address of the server in UDPClient.py. Next, you execute UDPServer.py, the compiled server program, in the server host. This creates a process in the server that idles until it is contacted by some client. Then you execute UDPClient.py, the compiled client program, in the client. This creates a process in the client. Finally, to use the application at the client, you type a sentence followed by a carriage return.

To develop your own UDP client-server application, you can begin by slightly modifying the client or server programs. For example, instead of converting all the letters to uppercase, the server could count the number of times the letter *s* appears and return this number. Or you can modify the client so that after receiving a capitalized sentence, the user can continue to send more sentences to the server.

2.7.2 Socket Programming with TCP

Unlike UDP, TCP is a connection-oriented protocol. This means that before the client and server can start to send data to each other, they first need to handshake and establish a TCP connection. One end of the TCP connection is attached to the client socket and the other end is attached to a server socket. When creating the TCP connection, we associate with it the client socket address (IP address and port number) and the server socket address (IP address and port number). With the TCP connection established, when one side wants to send data to the other side, it just drops the data into the TCP connection via its socket. This is different from UDP, for which the server must attach a destination address to the packet before dropping it into the socket.

Now let's take a closer look at the interaction of client and server programs in TCP. The client has the job of initiating contact with the server. In order for the server to be able to react to the client's initial contact, the server has to be ready. This implies two things. First, as in the case of UDP, the TCP server must be running as a process before the client attempts to initiate contact. Second, the server program must have a special door—more precisely, a special socket—that welcomes some initial contact from a client process running on an arbitrary host. Using our house/door analogy for a process/socket, we will sometimes refer to the client's initial contact as "knocking on the welcoming door."

With the server process running, the client process can initiate a TCP connection to the server. This is done in the client program by creating a TCP socket. When the client creates its TCP socket, it specifies the address of the welcoming socket in the server, namely, the IP address of the server host and the port number of the socket. After creating its socket, the client initiates a three-way handshake and establishes a TCP connection with the server. The three-way handshake, which takes place within the transport layer, is completely invisible to the client and server programs.

During the three-way handshake, the client process knocks on the welcoming door of the server process. When the server "hears" the knocking, it creates a new door—more precisely, a *new* socket that is dedicated to that particular client. In our example below, the welcoming door is a TCP socket object that we call `serverSocket`; the newly created socket dedicated to the client making the connection is called `connectionSocket`. Students who are encountering TCP sockets for the first time sometimes confuse the welcoming socket (which is the initial point of contact for all clients wanting to communicate with the server), and each newly created server-side connection socket that is subsequently created for communicating with each client.

From the application's perspective, the client's socket and the server's connection socket are directly connected by a pipe. As shown in Figure 2.29, the client process can send arbitrary bytes into its socket, and TCP guarantees that the server process will receive (through the connection socket) each byte in the order sent. TCP thus provides a reliable service between the client and server processes. Furthermore, just as people can go in and out the same door, the client process not only sends bytes

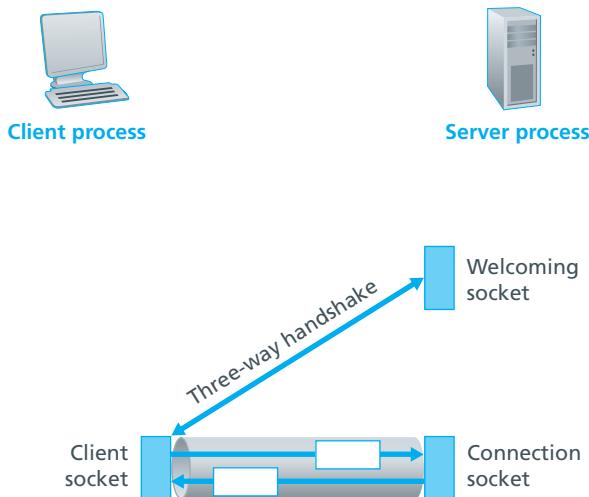


Figure 2.29 ♦ The TCP Server process has two sockets

into but also receives bytes from its socket; similarly, the server process not only receives bytes from but also sends bytes into its connection socket.

We use the same simple client-server application to demonstrate socket programming with TCP: The client sends one line of data to the server, the server capitalizes the line and sends it back to the client. Figure 2.30 highlights the main socket-related activity of the client and server that communicate over the TCP transport service.

TCPClient.py

Here is the code for the client side of the application:

```
from socket import *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = raw_input('Input lowercase sentence: ')
clientSocket.send(sentence)
modifiedSentence = clientSocket.recv(1024)
print 'From Server:', modifiedSentence
clientSocket.close()
```

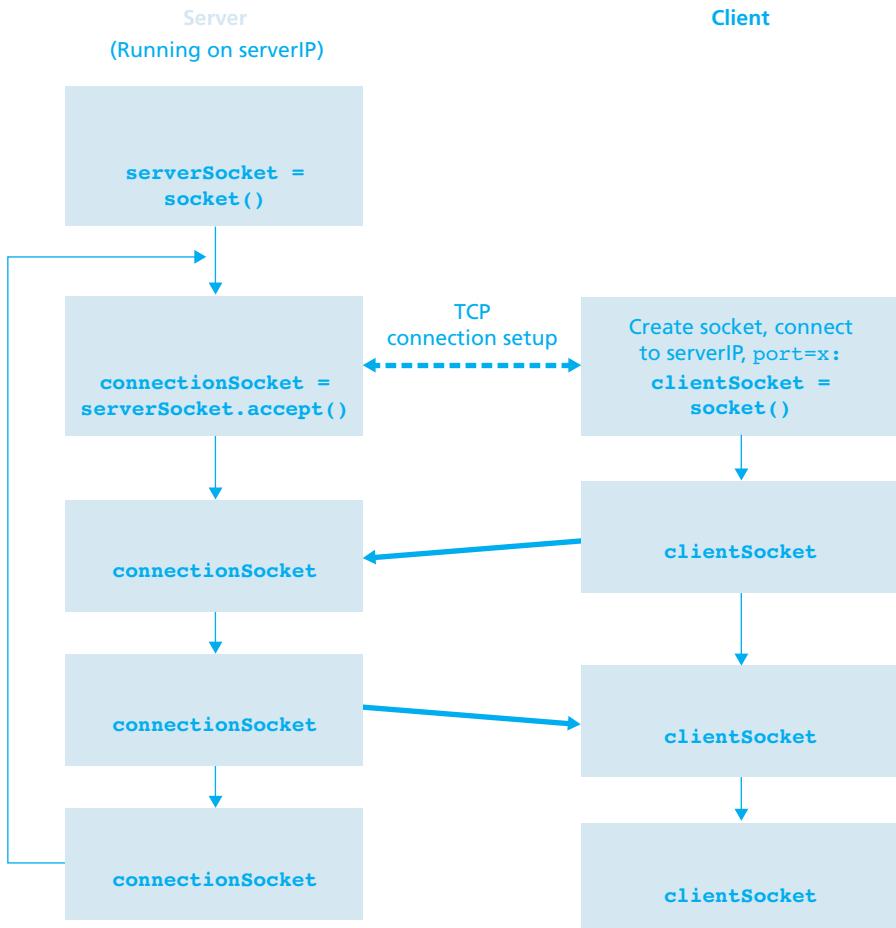


Figure 2.30 ♦ The client-server application using TCP

Let's now take a look at the various lines in the code that differ significantly from the UDP implementation. The first such line is the creation of the client socket.

```
clientSocket = socket(AF_INET, SOCK_STREAM)
```

This line creates the client's socket, called `clientSocket`. The first parameter again indicates that the underlying network is using IPv4. The second parameter indicates that the socket is of type `SOCK_STREAM`, which means it is a TCP socket (rather than a UDP socket). Note that we are again not specifying the port number.

of the client socket when we create it; we are instead letting the operating system do this for us. Now the next line of code is very different from what we saw in UDPClient:

```
clientSocket.connect((serverName, serverPort))
```

Recall that before the client can send data to the server (or vice versa) using a TCP socket, a TCP connection must first be established between the client and server. The above line initiates the TCP connection between the client and server. The parameter of the `connect()` method is the address of the server side of the connection. After this line of code is executed, the three-way handshake is performed and a TCP connection is established between the client and server.

```
sentence = raw_input('Input lowercase sentence:')
```

As with UDPClient, the above obtains a sentence from the user. The string `sentence` continues to gather characters until the user ends the line by typing a carriage return. The next line of code is also very different from UDPClient:

```
clientSocket.send(sentence)
```

The above line sends the string `sentence` through the client's socket and into the TCP connection. Note that the program does *not* explicitly create a packet and attach the destination address to the packet, as was the case with UDP sockets. Instead the client program simply drops the bytes in the string `sentence` into the TCP connection. The client then waits to receive bytes from the server.

```
modifiedSentence = clientSocket.recv(2048)
```

When characters arrive from the server, they get placed into the string `modifiedSentence`. Characters continue to accumulate in `modifiedSentence` until the line ends with a carriage return character. After printing the capitalized sentence, we close the client's socket:

```
clientSocket.close()
```

This last line closes the socket and, hence, closes the TCP connection between the client and the server. It causes TCP in the client to send a TCP message to TCP in the server (see Section 3.5).

TCPServer.py

Now let's take a look at the server program.

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
print 'The server is ready to receive'
while 1:
    connectionSocket, addr = serverSocket.accept()
    sentence = connectionSocket.recv(1024)
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence)
    connectionSocket.close()
```

Let's now take a look at the lines that differ significantly from UDPServer and TCP-Client. As with TCPClient, the server creates a TCP socket with:

```
serverSocket=socket(AF_INET,SOCK_STREAM)
```

Similar to UDPServer, we associate the server port number, `serverPort`, with this socket:

```
serverSocket.bind(('',serverPort))
```

But with TCP, `serverSocket` will be our welcoming socket. After establishing this welcoming door, we will wait and listen for some client to knock on the door:

```
serverSocket.listen(1)
```

This line has the server listen for TCP connection requests from the client. The parameter specifies the maximum number of queued connections (at least 1).

```
connectionSocket, addr = serverSocket.accept()
```

When a client knocks on this door, the program invokes the `accept()` method for `serverSocket`, which creates a new socket in the server, called `connectionSocket`, dedicated to this particular client. The client and server then complete the handshaking, creating a TCP connection between the client's `clientSocket` and the server's `connectionSocket`. With the TCP connection established, the client and server can now send bytes to each other over the connection. With TCP, all bytes sent from one side are not only guaranteed to arrive at the other side but also guaranteed arrive in order.

```
connectionSocket.close()
```

In this program, after sending the modified sentence to the client, we close the connection socket. But since `serverSocket` remains open, another client can now knock on the door and send the server a sentence to modify.

This completes our discussion of socket programming in TCP. You are encouraged to run the two programs in two separate hosts, and also to modify them to achieve slightly different goals. You should compare the UDP program pair with the TCP program pair and see how they differ. You should also do many of the socket programming assignments described at the ends of Chapters 2, 4, and 7. Finally, we hope someday, after mastering these and more advanced socket programs, you will write your own popular network application, become very rich and famous, and remember the authors of this textbook!

2.8 Summary

In this chapter, we've studied the conceptual and the implementation aspects of network applications. We've learned about the ubiquitous client-server architecture adopted by many Internet applications and seen its use in the HTTP, FTP, SMTP, POP3, and DNS protocols. We've studied these important application-level protocols, and their corresponding associated applications (the Web, file transfer, e-mail, and DNS) in some detail. We've also learned about the increasingly prevalent P2P architecture and how it is used in many applications. We've examined how the socket API can be used to build network applications. We've walked through the use of sockets for connection-oriented (TCP) and connectionless (UDP) end-to-end transport services. The first step in our journey down the layered network architecture is now complete!

At the very beginning of this book, in Section 1.1, we gave a rather vague, bare-bones definition of a protocol: “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 material in this chapter, and in particular our detailed study of the HTTP, FTP, SMTP, POP3, and DNS protocols, has now added considerable substance to this definition. Protocols are a key concept in networking; our study of application protocols has now given us the opportunity to develop a more intuitive feel for what protocols are all about.

In Section 2.1, we described the service models that TCP and UDP offer to applications that invoke them. We took an even closer look at these service models when we developed simple applications that run over TCP and UDP in Section 2.7. However, we have said little about how TCP and UDP provide these service models. For example, we know that TCP provides a reliable data service, but we haven't said yet how it does so. In the next chapter we'll take a careful look at not only the *what*, but also the *how* and *why* of transport protocols.

Equipped with knowledge about Internet application structure and application-level protocols, we're now ready to head further down the protocol stack and examine the transport layer in Chapter 3.



Homework Problems and Questions

Chapter 2 Review Questions

SECTION 2.1

- R1. List five nonproprietary Internet applications and the application-layer protocols that they use.
- R2. What is the difference between network architecture and application architecture?
- R3. For a communication session between a pair of processes, which process is the client and which is the server?
- R4. For a P2P file-sharing application, do you agree with the statement, “There is no notion of client and server sides of a communication session”? Why or why not?
- R5. What information is used by a process running on one host to identify a process running on another host?
- R6. Suppose you wanted to do a transaction from a remote client to a server as fast as possible. Would you use UDP or TCP? Why?
- R7. Referring to Figure 2.4, we see that none of the applications listed in Figure 2.4 requires both no data loss and timing. Can you conceive of an application that requires no data loss and that is also highly time-sensitive?
- R8. List the four broad classes of services that a transport protocol can provide. For each of the service classes, indicate if either UDP or TCP (or both) provides such a service.
- R9. Recall that TCP can be enhanced with SSL to provide process-to-process security services, including encryption. Does SSL operate at the transport layer or the application layer? If the application developer wants TCP to be enhanced with SSL, what does the developer have to do?

SECTIONS 2.2–2.5

- R10. What is meant by a handshaking protocol?
- R11. Why do HTTP, FTP, SMTP, and POP3 run on top of TCP rather than on UDP?
- R12. Consider an e-commerce site that wants to keep a purchase record for each of its customers. Describe how this can be done with cookies.

- R13. Describe how Web caching can reduce the delay in receiving a requested object. Will Web caching reduce the delay for all objects requested by a user or for only some of the objects? Why?
- R14. Telnet into a Web server and send a multiline request message. Include in the request message the `If-modified-since:` header line to force a response message with the `304 Not Modified` status code.
- R15. Why is it said that FTP sends control information “out-of-band”?
- R16. Suppose Alice, with a Web-based e-mail account (such as Hotmail or gmail), sends a message to Bob, who accesses his mail from his mail server using POP3. Discuss how the message gets from Alice’s host to Bob’s host. Be sure to list the series of application-layer protocols that are used to move the message between the two hosts.
- R17. Print out the header of an e-mail message you have recently received. How many `Received:` header lines are there? Analyze each of the header lines in the message.
- R18. From a user’s perspective, what is the difference between the download-and-delete mode and the download-and-keep mode in POP3?
- R19. Is it possible for an organization’s Web server and mail server to have exactly the same alias for a hostname (for example, `foo.com`)? What would be the type for the RR that contains the hostname of the mail server?
- R20. Look over your received emails, and examine the header of a message sent from a user with an .edu email address. Is it possible to determine from the header the IP address of the host from which the message was sent? Do the same for a message sent from a gmail account.

SECTION 2.6

- R21. In BitTorrent, suppose Alice provides chunks to Bob throughout a 30-second interval. Will Bob necessarily return the favor and provide chunks to Alice in this same interval? Why or why not?
- R22. Consider a new peer Alice that joins BitTorrent without possessing any chunks. Without any chunks, she cannot become a top-four uploader for any of the other peers, since she has nothing to upload. How then will Alice get her first chunk?
- R23. What is an overlay network? Does it include routers? What are the edges in the overlay network?
- R24. Consider a DHT with a mesh overlay topology (that is, every peer tracks all peers in the system). What are the advantages and disadvantages of such a design? What are the advantages and disadvantages of a circular DHT (with no shortcuts)?

R25. List at least four different applications that are naturally suitable for P2P architectures. (*Hint:* File distribution and instant messaging are two.)

SECTION 2.7

- R26. In Section 2.7, the UDP server described needed only one socket, whereas the TCP server needed two sockets. Why? If the TCP server were to support n simultaneous connections, each from a different client host, how many sockets would the TCP server need?
- R27. For the client-server application over TCP described in Section 2.7, why must the server program be executed before the client program? For the client-server application over UDP, why may the client program be executed before the server program?



Problems

P1. True or false?

- a. A user requests a Web page that consists of some text and three images. For this page, the client will send one request message and receive four response messages.
- b. Two distinct Web pages (for example, `www.mit.edu/research.html` and `www.mit.edu/students.html`) can be sent over the same persistent connection.
- c. With nonpersistent connections between browser and origin server, it is possible for a single TCP segment to carry two distinct HTTP request messages.
- d. The `Date:` header in the HTTP response message indicates when the object in the response was last modified.
- e. HTTP response messages never have an empty message body.

P2. Read RFC 959 for FTP. List all of the client commands that are supported by the RFC.

P3. Consider an HTTP client that wants to retrieve a Web document at a given URL. The IP address of the HTTP server is initially unknown. What transport and application-layer protocols besides HTTP are needed in this scenario?

P4. Consider the following string of ASCII characters that were captured by Wireshark when the browser sent an HTTP GET message (i.e., this is the actual content of an HTTP GET message). The characters `<cr><lf>` are carriage return and line-feed characters (that is, the italicized character string `<cr>` in the text below represents the single carriage-return character that was contained at that point in the HTTP header). Answer the following questions, indicating where in the HTTP GET message below you find the answer.

```
GET /cs453/index.html HTTP/1.1<cr><lf>Host: gai
a.cs.umass.edu<cr><lf>User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.2) Gecko/20040804 Netscape/7.2 (ax) <cr><lf>Accept: text/xml, application/xml, application/xhtml+xml, text/html;q=0.9, text/plain;q=0.8, image/png,*/*;q=0.5
<cr><lf>Accept-Language: en-us,en;q=0.5<cr><lf>Accept-Encoding: zip,deflate<cr><lf>Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7<cr><lf>Keep-Alive: 300<cr>
<lf>Connection:keep-alive<cr><lf><cr><lf>
```

- What is the URL of the document requested by the browser?
 - What version of HTTP is the browser running?
 - Does the browser request a non-persistent or a persistent connection?
 - What is the IP address of the host on which the browser is running?
 - What type of browser initiates this message? Why is the browser type needed in an HTTP request message?
- P5. The text below shows the reply sent from the server in response to the HTTP GET message in the question above. Answer the following questions, indicating where in the message below you find the answer.

```
HTTP/1.1 200 OK<cr><lf>Date: Tue, 07 Mar 2008
12:39:45GMT<cr><lf>Server: Apache/2.0.52 (Fedora)
<cr><lf>Last-Modified: Sat, 10 Dec 2005 18:27:46
GMT<cr><lf>ETag: "526c3-f22-a88a4c80"<cr><lf>Accept-
Ranges: bytes<cr><lf>Content-Length: 3874<cr><lf>
Keep-Alive: timeout=max=100<cr><lf>Connection:
Keep-Alive<cr><lf>Content-Type: text/html; charset=
ISO-8859-1<cr><lf><cr><lf><!DOCTYPE html public "-//w3c//dtd html 4.0 transitional//en"><lf><html><lf>
<head><lf> <meta http-equiv="Content-Type"
content="text/html; charset=iso-8859-1"><lf> <meta
name="GENERATOR" content="Mozilla/4.79 [en] (Windows NT
5.0; U) Netscape]"><lf> <title>CMPSCI 453 / 591 /
NTU-ST550A Spring 2005 homepage</title><lf></head><lf>
<much more document text following here (not shown)>
```

- Was the server able to successfully find the document or not? What time was the document reply provided?
- When was the document last modified?
- How many bytes are there in the document being returned?
- What are the first 5 bytes of the document being returned? Did the server agree to a persistent connection?

- P6. Obtain the HTTP/1.1 specification (RFC 2616). Answer the following questions:
- Explain the mechanism used for signaling between the client and server to indicate that a persistent connection is being closed. Can the client, the server, or both signal the close of a connection?
 - What encryption services are provided by HTTP?
 - Can a client open three or more simultaneous connections with a given server?
 - Either a server or a client may close a transport connection between them if either one detects the connection has been idle for some time. Is it possible that one side starts closing a connection while the other side is transmitting data via this connection? Explain.
- P7. Suppose within your Web browser you click on a link to obtain a Web page. The IP address for the associated URL is not cached in your local host, so a DNS lookup is necessary to obtain the IP address. Suppose that n DNS servers are visited before your host receives the IP address from DNS; the successive visits incur an RTT of $\text{RTT}_1, \dots, \text{RTT}_n$. Further suppose that the Web page associated with the link contains exactly one object, consisting of a small amount of HTML text. Let RTT_0 denote the RTT between the local host and the server containing the object. Assuming zero transmission time of the object, how much time elapses from when the client clicks on the link until the client receives the object?
- P8. Referring to Problem P7, suppose the HTML file references eight very small objects on the same server. Neglecting transmission times, how much time elapses with
- Non-persistent HTTP with no parallel TCP connections?
 - Non-persistent HTTP with the browser configured for 5 parallel connections?
 - Persistent HTTP?
- P9. Consider Figure 2.12, for which there is an institutional network connected to the Internet. Suppose that the average object size is 850,000 bits and that the average request rate from the institution's browsers to the origin servers is 16 requests per second. Also suppose that the amount of time it takes from when the router on the Internet side of the access link forwards an HTTP request until it receives the response is three seconds on average (see Section 2.2.5). Model the total average response time as the sum of the average access delay (that is, the delay from Internet router to institution router) and the average Internet delay. For the average access delay, use $\Delta/(1 - \Delta\beta)$, where Δ is the average time required to send an object over the access link and β is the arrival rate of objects to the access link.
- Find the total average response time.
 - Now suppose a cache is installed in the institutional LAN. Suppose the miss rate is 0.4. Find the total response time.

- P10. Consider a short, 10-meter link, over which a sender can transmit at a rate of 150 bits/sec in both directions. Suppose that packets containing data are 100,000 bits long, and packets containing only control (e.g., ACK or hand-shaking) are 200 bits long. Assume that N parallel connections each get $1/N$ of the link bandwidth. Now consider the HTTP protocol, and suppose that each downloaded object is 100 Kbytes long, and that the initial downloaded object contains 10 referenced objects from the same sender. Would parallel downloads via parallel instances of non-persistent HTTP make sense in this case? Now consider persistent HTTP. Do you expect significant gains over the non-persistent case? Justify and explain your answer.
- P11. Consider the scenario introduced in the previous problem. Now suppose that the link is shared by Bob with four other users. Bob uses parallel instances of non-persistent HTTP, and the other four users use non-persistent HTTP without parallel downloads.
- Do Bob's parallel connections help him get Web pages more quickly? Why or why not?
 - If all five users open five parallel instances of non-persistent HTTP, then would Bob's parallel connections still be beneficial? Why or why not?
- P12. Write a simple TCP program for a server that accepts lines of input from a client and prints the lines onto the server's standard output. (You can do this by modifying the TCPServer.py program in the text.) Compile and execute your program. On any other machine that contains a Web browser, set the proxy server in the browser to the host that is running your server program; also configure the port number appropriately. Your browser should now send its GET request messages to your server, and your server should display the messages on its standard output. Use this platform to determine whether your browser generates conditional GET messages for objects that are locally cached.
- P13. What is the difference between MAIL FROM: in SMTP and From: in the mail message itself?
- P14. How does SMTP mark the end of a message body? How about HTTP? Can HTTP use the same method as SMTP to mark the end of a message body? Explain.
- P15. Read RFC 5321 for SMTP. What does MTA stand for? Consider the following received spam email (modified from a real spam email). Assuming only the originator of this spam email is malicious and all other hosts are honest, identify the malicious host that has generated this spam email.

```
From - Fri Nov 07 13:41:30 2008
Return-Path: <tennis5@pp33head.com>
Received: from barmail.cs.umass.edu
(barmail.cs.umass.edu [128.119.240.3]) by cs.umass.edu
(8.13.1/8.12.6) for <hg@cs.umass.edu>; Fri, 7 Nov 2008
13:27:10 -0500
```

```
Received: from asusus-4b96 (localhost [127.0.0.1]) by
barmail.cs.umass.edu (Spam Firewall) for
<hg@cs.umass.edu>; Fri, 7 Nov 2008 13:27:07 -0500
(EST)
Received: from asusus-4b96 ([58.88.21.177]) by
barmail.cs.umass.edu for <hg@cs.umass.edu>; Fri,
07 Nov 2008 13:27:07 -0500 (EST)
Received: from [58.88.21.177] by
inbnd55.exchangeddd.com; Sat, 8 Nov 2008 01:27:07 +0700
From: "Jonny" <tennis5@pp33head.com>
To: <hg@cs.umass.edu>
Subject: How to secure your savings
```

P16. Read the POP3 RFC, RFC 1939. What is the purpose of the UIDL POP3 command?

P17. Consider accessing your e-mail with POP3.

- Suppose you have configured your POP mail client to operate in the download-and-delete mode. Complete the following transaction:

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: blah blah ...
S: .....blah
S: .
?
?
```

- Suppose you have configured your POP mail client to operate in the download-and-keep mode. Complete the following transaction:

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: blah blah ...
S: .....blah
S: .
?
?
```

- c. Suppose you have configured your POP mail client to operate in the download-and-keep mode. Using your transcript in part (b), suppose you retrieve messages 1 and 2, exit POP, and then five minutes later you again access POP to retrieve new e-mail. Suppose that in the five-minute interval no new messages have been sent to you. Provide a transcript of this second POP session.

- P18. a. What is a *whois* database?
- b. Use various whois databases on the Internet to obtain the names of two DNS servers. Indicate which whois databases you used.
 - c. Use nslookup on your local host to send DNS queries to three DNS servers: your local DNS server and the two DNS servers you found in part (b). Try querying for Type A, NS, and MX reports. Summarize your findings.
 - d. Use nslookup to find a Web server that has multiple IP addresses. Does the Web server of your institution (school or company) have multiple IP addresses?
 - e. Use the ARIN whois database to determine the IP address range used by your university.
 - f. Describe how an attacker can use whois databases and the nslookup tool to perform reconnaissance on an institution before launching an attack.
 - g. Discuss why whois databases should be publicly available.
- P19. In this problem, we use the useful *dig* tool available on Unix and Linux hosts to explore the hierarchy of DNS servers. Recall that in Figure 2.21, a DNS server higher in the DNS hierarchy delegates a DNS query to a DNS server lower in the hierarchy, by sending back to the DNS client the name of that lower-level DNS server. First read the man page for *dig*, and then answer the following questions.
- a. Starting with a root DNS server (from one of the root servers [a-m].root-servers.net), initiate a sequence of queries for the IP address for your department's Web server by using *dig*. Show the list of the names of DNS servers in the delegation chain in answering your query.
 - b. Repeat part a) for several popular Web sites, such as google.com, yahoo.com, or amazon.com.
- P20. Suppose you can access the caches in the local DNS servers of your department. Can you propose a way to roughly determine the Web servers (outside your department) that are most popular among the users in your department? Explain.
- P21. Suppose that your department has a local DNS server for all computers in the department. You are an ordinary user (i.e., not a network/system administrator). Can you determine if an external Web site was likely accessed from a computer in your department a couple of seconds ago? Explain.

- P22. Consider distributing a file of $F = 15$ Gbits to N peers. The server has an upload rate of $u_s = 30$ Mbps, and each peer has a download rate of $d_i = 2$ Mbps and an upload rate of u . For $N = 10, 100,$ and $1,000$ and $u = 300$ Kbps, 700 Kbps, and 2 Mbps, prepare a chart giving the minimum distribution time for each of the combinations of N and u for both client-server distribution and P2P distribution.
- P23. Consider distributing a file of F bits to N peers using a client-server architecture. Assume a fluid model where the server can simultaneously transmit to multiple peers, transmitting to each peer at different rates, as long as the combined rate does not exceed u_s .
- Suppose that $u_s/N \leq d_{\min}$. Specify a distribution scheme that has a distribution time of NF/u_s .
 - Suppose that $u_s/N \geq d_{\min}$. Specify a distribution scheme that has a distribution time of F/d_{\min} .
 - Conclude that the minimum distribution time is in general given by $\max\{NF/u_s, F/d_{\min}\}$.
- P24. Consider distributing a file of F bits to N peers using a P2P architecture. Assume a fluid model. For simplicity assume that d_{\min} is very large, so that peer download bandwidth is never a bottleneck.
- Suppose that $u_s \leq (u_s + u_1 + \dots + u_N)/N$. Specify a distribution scheme that has a distribution time of F/u_s .
 - Suppose that $u_s \geq (u_s + u_1 + \dots + u_N)/N$. Specify a distribution scheme that has a distribution time of $NF/(u_s + u_1 + \dots + u_N)$.
 - Conclude that the minimum distribution time is in general given by $\max\{F/u_s, NF/(u_s + u_1 + \dots + u_N)\}$.
- P25. Consider an overlay network with N active peers, with each pair of peers having an active TCP connection. Additionally, suppose that the TCP connections pass through a total of M routers. How many nodes and edges are there in the corresponding overlay network?
- P26. Suppose Bob joins a BitTorrent torrent, but he does not want to upload any data to any other peers (so called free-riding).
- Bob claims that he can receive a complete copy of the file that is shared by the swarm. Is Bob's claim possible? Why or why not?
 - Bob further claims that he can further make his "free-riding" more efficient by using a collection of multiple computers (with distinct IP addresses) in the computer lab in his department. How can he do that?
- P27. In the circular DHT example in Section 2.6.2, suppose that peer 3 learns that peer 5 has left. How does peer 3 update its successor state information? Which peer is now its first successor? Its second successor?

- P28. In the circular DHT example in Section 2.6.2, suppose that a new peer 6 wants to join the DHT and peer 6 initially only knows peer 15's IP address. What steps are taken?
- P29. Because an integer in $[0, 2^n - 1]$ can be expressed as an n -bit binary number in a DHT, each key can be expressed as $k = (k_0, k_1, \dots, k_{n-1})$, and each peer identifier can be expressed $p = (p_0, p_1, \dots, p_{n-1})$. Let's now define the XOR distance between a key k and peer p as

$$d(k, p) = \sum_{j=0}^{n-1} |k_j - p_j| 2^j$$

Describe how this metric can be used to assign (key, value) pairs to peers. (To learn about how to build an efficient DHT using this natural metric, see [Maymounkov 2002] in which the Kademlia DHT is described.)

- P30. As DHTs are overlay networks, they may not necessarily match the underlay physical network well in the sense that two neighboring peers might be physically very far away; for example, one peer could be in Asia and its neighbor could be in North America. If we randomly and uniformly assign identifiers to newly joined peers, would this assignment scheme cause such a mismatch? Explain. And how would such a mismatch affect the DHT's performance?
- P31. Install and compile the Python programs TCPClient and UDPClient on one host and TCPServer and UDPServer on another host.
- Suppose you run TCPClient before you run TCPServer. What happens? Why?
 - Suppose you run UDPClient before you run UDPServer. What happens? Why?
 - What happens if you use different port numbers for the client and server sides?
- P32. Suppose that in UDPClient.py, after we create the socket, we add the line:

```
clientSocket.bind(('', 5432))
```

Will it become necessary to change UDPServer.py? What are the port numbers for the sockets in UDPClient and UDPServer? What were they before making this change?

- P33. Can you configure your browser to open multiple simultaneous connections to a Web site? What are the advantages and disadvantages of having a large number of simultaneous TCP connections?
- P34. We have seen that Internet TCP sockets treat the data being sent as a byte stream but UDP sockets recognize message boundaries. What are one

- advantage and one disadvantage of byte-oriented API versus having the API explicitly recognize and preserve application-defined message boundaries?
- P35. What is the Apache Web server? How much does it cost? What functionality does it currently have? You may want to look at Wikipedia to answer this question.
- P36. Many BitTorrent clients use DHTs to create a distributed tracker. For these DHTs, what is the “key” and what is the “value”?



Socket Programming Assignments

The companion Web site includes six socket programming assignments. The first four assignments are summarized below. The fifth assignment makes use of the ICMP protocol and is summarized at the end of Chapter 4. The sixth assignment employs multimedia protocols and is summarized at the end of Chapter 7. It is highly recommended that students complete several, if not all, of these assignments. Students can find full details of these assignments, as well as important snippets of the Python code, at the Web site <http://www.awl.com/kurose-ross>.

Assignment 1: Web Server

In this assignment, you will develop a simple Web server in Python that is capable of processing only one request. Specifically, your Web server will (i) create a connection socket when contacted by a client (browser); (ii) receive the HTTP request from this connection; (iii) parse the request to determine the specific file being requested; (iv) get the requested file from the server’s file system; (v) create an HTTP response message consisting of the requested file preceded by header lines; and (vi) send the response over the TCP connection to the requesting browser. If a browser requests a file that is not present in your server, your server should return a “404 Not Found” error message.

In the companion Web site, we provide the skeleton code for your server. Your job is to complete the code, run your server, and then test your server by sending requests from browsers running on different hosts. If you run your server on a host that already has a Web server running on it, then you should use a different port than port 80 for your Web server.

Assignment 2: UDP Pinger

In this programming assignment, you will write a client ping program in Python. Your client will send a simple ping message to a server, receive a corresponding pong message back from the server, and determine the delay between when the client sent the ping message and received the pong message. This delay is called the Round Trip Time (RTT). The functionality provided by the client and server is

similar to the functionality provided by standard ping program available in modern operating systems. However, standard ping programs use the Internet Control Message Protocol (ICMP) (which we will study in Chapter 4). Here we will create a nonstandard (but simple!) UDP-based ping program.

Your ping program is to send 10 ping messages to the target server over UDP. For each message, your client is to determine and print the RTT when the corresponding pong message is returned. Because UDP is an unreliable protocol, a packet sent by the client or server may be lost. For this reason, the client cannot wait indefinitely for a reply to a ping message. You should have the client wait up to one second for a reply from the server; if no reply is received, the client should assume that the packet was lost and print a message accordingly.

In this assignment, you will be given the complete code for the server (available in the companion Web site). Your job is to write the client code, which will be very similar to the server code. It is recommended that you first study carefully the server code. You can then write your client code, liberally cutting and pasting lines from the server code.

Assignment 3: Mail Client

The goal of this programming assignment is to create a simple mail client that sends email to any recipient. Your client will need to establish a TCP connection with a mail server (e.g., a Google mail server), dialogue with the mail server using the SMTP protocol, send an email message to a recipient (e.g., your friend) via the mail server, and finally close the TCP connection with the mail server.

For this assignment, the companion Web site provides the skeleton code for your client. Your job is to complete the code and test your client by sending email to different user accounts. You may also try sending through different servers (for example, through a Google mail server and through your university mail server).

Assignment 4: Multi-Threaded Web Proxy

In this assignment, you will develop a Web proxy. When your proxy receives an HTTP request for an object from a browser, it generates a new HTTP request for the same object and sends it to the origin server. When the proxy receives the corresponding HTTP response with the object from the origin server, it creates a new HTTP response, including the object, and sends it to the client. This proxy will be multi-threaded, so that it will be able to handle multiple requests at the same time.

For this assignment, the companion Web site provides the skeleton code for the proxy server. Your job is to complete the code, and then test it by having different browsers request Web objects via your proxy.



Wireshark Lab: HTTP

Having gotten our feet wet with the Wireshark packet sniffer in Lab 1, we're now ready to use Wireshark to investigate protocols in operation. In this lab, we'll explore several aspects of the HTTP protocol: the basic GET/reply interaction, HTTP message formats, retrieving large HTML files, retrieving HTML files with embedded URLs, persistent and non-persistent connections, and HTTP authentication and security.

As is the case with all Wireshark labs, the full description of this lab is available at this book's Web site, <http://www.awl.com/kurose-ross>.



VideoNote
Using Wireshark to
investigate the
HTTP protocol



Wireshark Lab: DNS

In this lab, we take a closer look at the client side of the DNS, the protocol that translates Internet hostnames to IP addresses. Recall from Section 2.5 that the client's role in the DNS is relatively simple—a client sends a query to its local DNS server and receives a response back. Much can go on under the covers, invisible to the DNS clients, as the hierarchical DNS servers communicate with each other to either recursively or iteratively resolve the client's DNS query. From the DNS client's standpoint, however, the protocol is quite simple—a query is formulated to the local DNS server and a response is received from that server. We observe DNS in action in this lab.

As is the case with all Wireshark labs, the full description of this lab is available at this book's Web site, <http://www.awl.com/kurose-ross>.

AN INTERVIEW WITH...

Marc Andreessen

Marc Andreessen is the co-creator of Mosaic, the Web browser that popularized the World Wide Web in 1993. Mosaic had a clean, easily understood interface and was the first browser to display images in-line with text. In 1994, Marc Andreessen and Jim Clark founded Netscape, whose browser was by far the most popular browser through the mid-1990s. Netscape also developed the Secure Sockets Layer (SSL) protocol and many Internet server products, including mail servers and SSL-based Web servers. He is now a co-founder and general partner of venture capital firm Andreessen Horowitz, overseeing portfolio development with holdings that include Facebook, Foursquare, Groupon, Jawbone, Twitter, and Zynga. He serves on numerous boards, including Bump, eBay, Glam Media, Facebook, and Hewlett-Packard. He holds a BS in Computer Science from the University of Illinois at Urbana-Champaign.



How did you become interested in computing? Did you always know that you wanted to work in information technology?

The video game and personal computing revolutions hit right when I was growing up—personal computing was the new technology frontier in the late 70's and early 80's. And it wasn't just Apple and the IBM PC, but hundreds of new companies like Commodore and Atari as well. I taught myself to program out of a book called "Instant Freeze-Dried BASIC" at age 10, and got my first computer (a TRS-80 Color Computer—look it up!) at age 12.

Please describe one or two of the most exciting projects you have worked on during your career. What were the biggest challenges?

Undoubtedly the most exciting project was the original Mosaic web browser in '92-'93—and the biggest challenge was getting anyone to take it seriously back then. At the time, everyone thought the interactive future would be delivered as "interactive television" by huge companies, not as the Internet by startups.

What excites you about the future of networking and the Internet? What are your biggest concerns?

The most exciting thing is the huge unexplored frontier of applications and services that programmers and entrepreneurs are able to explore—the Internet has unleashed creativity at

a level that I don't think we've ever seen before. My biggest concern is the principle of unintended consequences—we don't always know the implications of what we do, such as the Internet being used by governments to run a new level of surveillance on citizens.

Is there anything in particular students should be aware of as Web technology advances?

The rate of change—the most important thing to learn is how to learn—how to flexibly adapt to changes in the specific technologies, and how to keep an open mind on the new opportunities and possibilities as you move through your career.

What people inspired you professionally?

Vannevar Bush, Ted Nelson, Doug Engelbart, Nolan Bushnell, Bill Hewlett and Dave Packard, Ken Olsen, Steve Jobs, Steve Wozniak, Andy Grove, Grace Hopper, Hedy Lamarr, Alan Turing, Richard Stallman.

What are your recommendations for students who want to pursue careers in computing and information technology?

Go as deep as you possibly can on understanding how technology is created, and then complement with learning how business works.

Can technology solve the world's problems?

No, but we advance the standard of living of people through economic growth, and most economic growth throughout history has come from technology—so that's as good as it gets.

This page intentionally left blank

Transport Layer



Residing between the application and network layers, the transport layer is a central piece of the layered network architecture. It has the critical role of providing communication services directly to the application processes running on different hosts. The pedagogic approach we take in this chapter is to alternate between discussions of transport-layer principles and discussions of how these principles are implemented in existing protocols; as usual, particular emphasis will be given to Internet protocols, in particular the TCP and UDP transport-layer protocols.

We'll begin by discussing the relationship between the transport and network layers. This sets the stage for examining the first critical function of the transport layer—extending the network layer's delivery service between two end systems to a delivery service between two application-layer processes running on the end systems. We'll illustrate this function in our coverage of the Internet's connectionless transport protocol, UDP.

We'll then return to principles and confront one of the most fundamental problems in computer networking—how two entities can communicate reliably over a medium that may lose and corrupt data. Through a series of increasingly complicated (and realistic!) scenarios, we'll build up an array of techniques that transport protocols use to solve this problem. We'll then show how these principles are embodied in TCP, the Internet's connection-oriented transport protocol.

We'll next move on to a second fundamentally important problem in networking—controlling the transmission rate of transport-layer entities in order to avoid, or

recover from, congestion within the network. We'll consider the causes and consequences of congestion, as well as commonly used congestion-control techniques. After obtaining a solid understanding of the issues behind congestion control, we'll study TCP's approach to congestion control.

3.1 Introduction and Transport-Layer Services

In the previous two chapters we touched on the role of the transport layer and the services that it provides. Let's quickly review what we have already learned about the transport layer.

A transport-layer protocol provides for **logical communication** between application processes running on different hosts. By *logical communication*, we mean that from an application's perspective, it is as if the hosts running the processes were directly connected; in reality, the hosts may be on opposite sides of the planet, connected via numerous routers and a wide range of link types. Application processes use the logical communication provided by the transport layer to send messages to each other, free from the worry of the details of the physical infrastructure used to carry these messages. Figure 3.1 illustrates the notion of logical communication.

As shown in Figure 3.1, transport-layer protocols are implemented in the end systems but not in network routers. On the sending side, the transport layer converts the application-layer messages it receives from a sending application process into transport-layer packets, known as transport-layer **segments** in Internet terminology. This is done by (possibly) breaking the application messages into smaller chunks and adding a transport-layer header to each chunk to create the transport-layer segment. The transport layer then passes the segment to the network layer at the sending end system, where the segment is encapsulated within a network-layer packet (a datagram) and sent to the destination. It's important to note that network routers act only on the network-layer fields of the datagram; that is, they do not examine the fields of the transport-layer segment encapsulated with the datagram. On the receiving side, the network layer extracts the transport-layer segment from the datagram and passes the segment up to the transport layer. The transport layer then processes the received segment, making the data in the segment available to the receiving application.

More than one transport-layer protocol may be available to network applications. For example, the Internet has two protocols—TCP and UDP. Each of these protocols provides a different set of transport-layer services to the invoking application.

3.1.1 Relationship Between Transport and Network Layers

Recall that the transport layer lies just above the network layer in the protocol stack. Whereas a transport-layer protocol provides logical communication between *processes* running on different hosts, a network-layer protocol provides logical

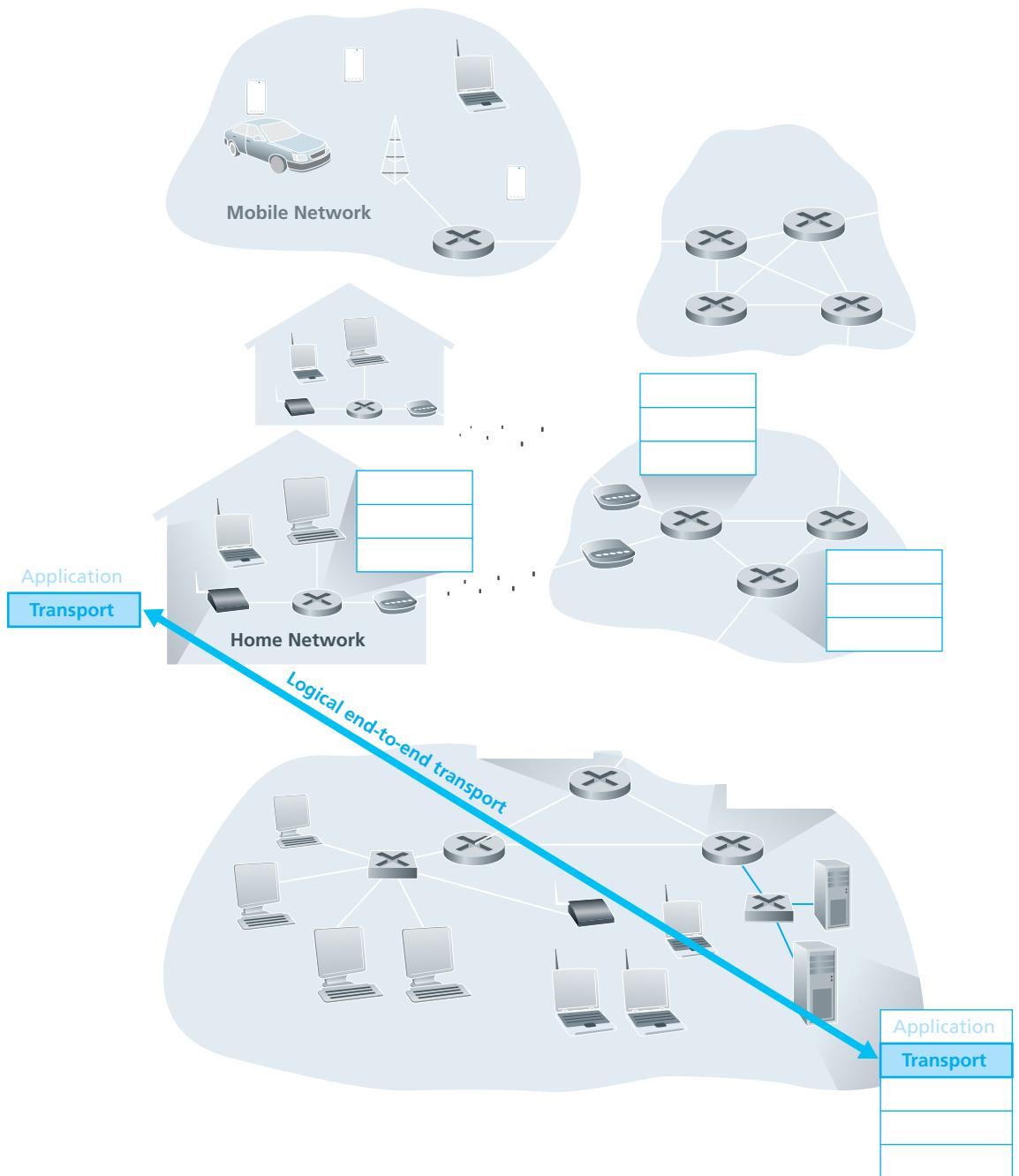


Figure 3.1 ♦ The transport layer provides logical rather than physical communication between application processes

communication between *hosts*. This distinction is subtle but important. Let's examine this distinction with the aid of a household analogy.

Consider two houses, one on the East Coast and the other on the West Coast, with each house being home to a dozen kids. The kids in the East Coast household are cousins of the kids in the West Coast household. The kids in the two households love to write to each other—each kid writes each cousin every week, with each letter delivered by the traditional postal service in a separate envelope. Thus, each household sends 144 letters to the other household every week. (These kids would save a lot of money if they had e-mail!) In each of the households there is one kid—Ann in the West Coast house and Bill in the East Coast house—responsible for mail collection and mail distribution. Each week Ann visits all her brothers and sisters, collects the mail, and gives the mail to a postal-service mail carrier, who makes daily visits to the house. When letters arrive at the West Coast house, Ann also has the job of distributing the mail to her brothers and sisters. Bill has a similar job on the East Coast.

In this example, the postal service provides logical communication between the two houses—the postal service moves mail from house to house, not from person to person. On the other hand, Ann and Bill provide logical communication among the cousins—Ann and Bill pick up mail from, and deliver mail to, their brothers and sisters. Note that from the cousins' perspective, Ann and Bill *are* the mail service, even though Ann and Bill are only a part (the end-system part) of the end-to-end delivery process. This household example serves as a nice analogy for explaining how the transport layer relates to the network layer:

- application messages = letters in envelopes
- processes = cousins
- hosts (also called end systems) = houses
- transport-layer protocol = Ann and Bill
- network-layer protocol = postal service (including mail carriers)

Continuing with this analogy, note that Ann and Bill do all their work within their respective homes; they are not involved, for example, in sorting mail in any intermediate mail center or in moving mail from one mail center to another. Similarly, transport-layer protocols live in the end systems. Within an end system, a transport protocol moves messages from application processes to the network edge (that is, the network layer) and vice versa, but it doesn't have any say about how the messages are moved within the network core. In fact, as illustrated in Figure 3.1, intermediate routers neither act on, nor recognize, any information that the transport layer may have added to the application messages.

Continuing with our family saga, suppose now that when Ann and Bill go on vacation, another cousin pair—say, Susan and Harvey—substitute for them and provide the household-internal collection and delivery of mail. Unfortunately for the two families, Susan and Harvey do not do the collection and delivery in exactly the same way as Ann and Bill. Being younger kids, Susan and Harvey pick up and drop off the mail less frequently and occasionally lose letters (which are sometimes

chewed up by the family dog). Thus, the cousin-pair Susan and Harvey do not provide the same set of services (that is, the same service model) as Ann and Bill. In an analogous manner, a computer network may make available multiple transport protocols, with each protocol offering a different service model to applications.

The possible services that Ann and Bill can provide are clearly constrained by the possible services that the postal service provides. For example, if the postal service doesn't provide a maximum bound on how long it can take to deliver mail between the two houses (for example, three days), then there is no way that Ann and Bill can guarantee a maximum delay for mail delivery between any of the cousin pairs. In a similar manner, the services that a transport protocol can provide are often constrained by the service model of the underlying network-layer protocol. If the network-layer protocol cannot provide delay or bandwidth guarantees for transport-layer segments sent between hosts, then the transport-layer protocol cannot provide delay or bandwidth guarantees for application messages sent between processes.

Nevertheless, certain services *can* be offered by a transport protocol even when the underlying network protocol doesn't offer the corresponding service at the network layer. For example, as we'll see in this chapter, a transport protocol can offer reliable data transfer service to an application even when the underlying network protocol is unreliable, that is, even when the network protocol loses, garbles, or duplicates packets. As another example (which we'll explore in Chapter 8 when we discuss network security), a transport protocol can use encryption to guarantee that application messages are not read by intruders, even when the network layer cannot guarantee the confidentiality of transport-layer segments.

3.1.2 Overview of the Transport Layer in the Internet

Recall that the Internet, and more generally a TCP/IP network, makes two distinct transport-layer protocols available to the application layer. One of these protocols is **UDP** (User Datagram Protocol), which provides an unreliable, connectionless service to the invoking application. The second of these protocols is **TCP** (Transmission Control Protocol), which provides a reliable, connection-oriented service to the invoking application. When designing a network application, the application developer must specify one of these two transport protocols. As we saw in Section 2.7, the application developer selects between UDP and TCP when creating sockets.

To simplify terminology, when in an Internet context, we refer to the transport-layer packet as a *segment*. We mention, however, that the Internet literature (for example, the RFCs) also refers to the transport-layer packet for TCP as a segment but often refers to the packet for UDP as a datagram. But this same Internet literature also uses the term *datagram* for the network-layer packet! For an introductory book on computer networking such as this, we believe that it is less confusing to refer to both TCP and UDP packets as segments, and reserve the term *datagram* for the network-layer packet.

Before proceeding with our brief introduction of UDP and TCP, it will be useful to say a few words about the Internet's network layer. (We'll learn about the network layer in detail in Chapter 4.) The Internet's network-layer protocol has a

name—IP, for Internet Protocol. IP provides logical communication between hosts. The IP service model is a **best-effort delivery service**. This means that IP makes its “best effort” to deliver segments between communicating hosts, *but it makes no guarantees*. In particular, it does not guarantee segment delivery, it does not guarantee orderly delivery of segments, and it does not guarantee the integrity of the data in the segments. For these reasons, IP is said to be an **unreliable service**. We also mention here that every host has at least one network-layer address, a so-called IP address. We’ll examine IP addressing in detail in Chapter 4; for this chapter we need only keep in mind that *each host has an IP address*.

Having taken a glimpse at the IP service model, let’s now summarize the service models provided by UDP and TCP. The most fundamental responsibility of UDP and TCP is to extend IP’s delivery service between two end systems to a delivery service between two processes running on the end systems. Extending host-to-host delivery to process-to-process delivery is called **transport-layer multiplexing** and **demultiplexing**. We’ll discuss transport-layer multiplexing and demultiplexing in the next section. UDP and TCP also provide integrity checking by including error-detection fields in their segments’ headers. These two minimal transport-layer services—process-to-process data delivery and error checking—are the only two services that UDP provides! In particular, like IP, UDP is an unreliable service—it does not guarantee that data sent by one process will arrive intact (or at all!) to the destination process. UDP is discussed in detail in Section 3.3.

TCP, on the other hand, offers several additional services to applications. First and foremost, it provides **reliable data transfer**. Using flow control, sequence numbers, acknowledgments, and timers (techniques we’ll explore in detail in this chapter), TCP ensures that data is delivered from sending process to receiving process, correctly and in order. TCP thus converts IP’s unreliable service between end systems into a reliable data transport service between processes. TCP also provides **congestion control**. Congestion control is not so much a service provided to the invoking application as it is a service for the Internet as a whole, a service for the general good. Loosely speaking, TCP congestion control prevents any one TCP connection from swamping the links and routers between communicating hosts with an excessive amount of traffic. TCP strives to give each connection traversing a congested link an equal share of the link bandwidth. This is done by regulating the rate at which the sending sides of TCP connections can send traffic into the network. UDP traffic, on the other hand, is unregulated. An application using UDP transport can send at any rate it pleases, for as long as it pleases.

A protocol that provides reliable data transfer and congestion control is necessarily complex. We’ll need several sections to cover the principles of reliable data transfer and congestion control, and additional sections to cover the TCP protocol itself. These topics are investigated in Sections 3.4 through 3.8. The approach taken in this chapter is to alternate between basic principles and the TCP protocol. For example, we’ll first discuss reliable data transfer in a general setting and then discuss how TCP specifically provides reliable data transfer. Similarly, we’ll first

discuss congestion control in a general setting and then discuss how TCP performs congestion control. But before getting into all this good stuff, let's first look at transport-layer multiplexing and demultiplexing.

3.2 Multiplexing and Demultiplexing

In this section, we discuss transport-layer multiplexing and demultiplexing, that is, extending the host-to-host delivery service provided by the network layer to a process-to-process delivery service for applications running on the hosts. In order to keep the discussion concrete, we'll discuss this basic transport-layer service in the context of the Internet. We emphasize, however, that a multiplexing/demultiplexing service is needed for all computer networks.

At the destination host, the transport layer receives segments from the network layer just below. The transport layer has the responsibility of delivering the data in these segments to the appropriate application process running in the host. Let's take a look at an example. Suppose you are sitting in front of your computer, and you are downloading Web pages while running one FTP session and two Telnet sessions. You therefore have four network application processes running—two Telnet processes, one FTP process, and one HTTP process. When the transport layer in your computer receives data from the network layer below, it needs to direct the received data to one of these four processes. Let's now examine how this is done.

First recall from Section 2.7 that a process (as part of a network application) can have one or more **sockets**, doors through which data passes from the network to the process and through which data passes from the process to the network. Thus, as shown in Figure 3.2, the transport layer in the receiving host does not actually deliver data directly to a process, but instead to an intermediary socket. Because at any given time there can be more than one socket in the receiving host, each socket has a unique identifier. The format of the identifier depends on whether the socket is a UDP or a TCP socket, as we'll discuss shortly.

Now let's consider how a receiving host directs an incoming transport-layer segment to the appropriate socket. Each transport-layer segment has a set of fields in the segment for this purpose. At the receiving end, the transport layer examines these fields to identify the receiving socket and then directs the segment to that socket. This job of delivering the data in a transport-layer segment to the correct socket is called **demultiplexing**. The job of gathering data chunks at the source host from different sockets, encapsulating each data chunk with header information (that will later be used in demultiplexing) to create segments, and passing the segments to the network layer is called **multiplexing**. Note that the transport layer in the middle host in Figure 3.2 must demultiplex segments arriving from the network layer below to either process P_1 or P_2 above; this is done by directing the arriving segment's data to the corresponding process's socket. The transport layer in the middle host must also

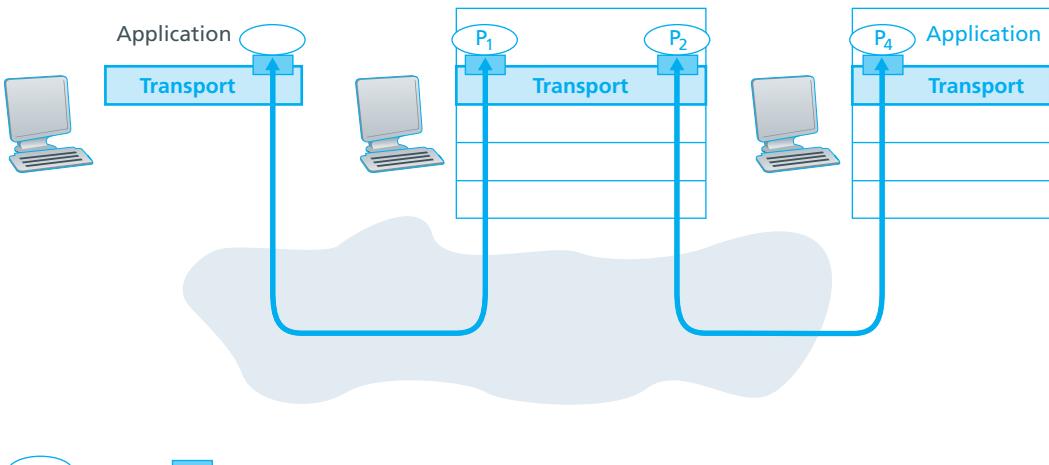


Figure 3.2 ♦ Transport-layer multiplexing and demultiplexing

gather outgoing data from these sockets, form transport-layer segments, and pass these segments down to the network layer. Although we have introduced multiplexing and demultiplexing in the context of the Internet transport protocols, it's important to realize that they are concerns whenever a single protocol at one layer (at the transport layer or elsewhere) is used by multiple protocols at the next higher layer.

To illustrate the demultiplexing job, recall the household analogy in the previous section. Each of the kids is identified by his or her name. When Bill receives a batch of mail from the mail carrier, he performs a demultiplexing operation by observing to whom the letters are addressed and then hand delivering the mail to his brothers and sisters. Ann performs a multiplexing operation when she collects letters from her brothers and sisters and gives the collected mail to the mail person.

Now that we understand the roles of transport-layer multiplexing and demultiplexing, let us examine how it is actually done in a host. From the discussion above, we know that transport-layer multiplexing requires (1) that sockets have unique identifiers, and (2) that each segment have special fields that indicate the socket to which the segment is to be delivered. These special fields, illustrated in Figure 3.3, are the **source port number field** and the **destination port number field**. (The UDP and TCP segments have other fields as well, as discussed in the subsequent sections of this chapter.) Each port number is a 16-bit number, ranging from 0 to 65535. The port numbers ranging from 0 to 1023 are called **well-known port numbers** and are restricted, which means that they are reserved for use by well-known application protocols such as HTTP (which uses port number 80) and FTP (which uses port number 21). The list of well-known port numbers is given in RFC 1700 and is updated at <http://www.iana.org> [RFC 3232]. When we develop a new

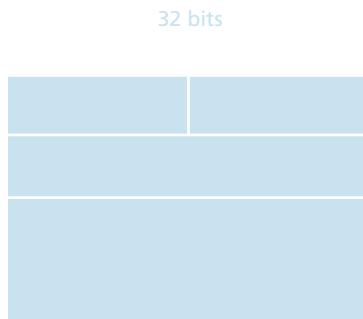


Figure 3.3 ♦ Source and destination port-number fields in a transport-layer segment

application (such as the simple application developed in Section 2.7), we must assign the application a port number.

It should now be clear how the transport layer *could* implement the demultiplexing service: Each socket in the host could be assigned a port number, and when a segment arrives at the host, the transport layer examines the destination port number in the segment and directs the segment to the corresponding socket. The segment's data then passes through the socket into the attached process. As we'll see, this is basically how UDP does it. However, we'll also see that multiplexing/demultiplexing in TCP is yet more subtle.

Connectionless Multiplexing and Demultiplexing

Recall from Section 2.7.1 that the Python program running in a host can create a UDP socket with the line

```
clientSocket = socket(socket.AF_INET, socket.SOCK_DGRAM)
```

When a UDP socket is created in this manner, the transport layer automatically assigns a port number to the socket. In particular, the transport layer assigns a port number in the range 1024 to 65535 that is currently not being used by any other UDP port in the host. Alternatively, we can add a line into our Python program after we create the socket to associate a specific port number (say, 19157) to this UDP socket via the socket `bind()` method:

```
clientSocket.bind(('', 19157))
```

If the application developer writing the code were implementing the server side of a “well-known protocol,” then the developer would have to assign the corresponding

well-known port number. Typically, the client side of the application lets the transport layer automatically (and transparently) assign the port number, whereas the server side of the application assigns a specific port number.

With port numbers assigned to UDP sockets, we can now precisely describe UDP multiplexing/demultiplexing. Suppose a process in Host A, with UDP port 19157, wants to send a chunk of application data to a process with UDP port 46428 in Host B. The transport layer in Host A creates a transport-layer segment that includes the application data, the source port number (19157), the destination port number (46428), and two other values (which will be discussed later, but are unimportant for the current discussion). The transport layer then passes the resulting segment to the network layer. The network layer encapsulates the segment in an IP datagram and makes a best-effort attempt to deliver the segment to the receiving host. If the segment arrives at the receiving Host B, the transport layer at the receiving host examines the destination port number in the segment (46428) and delivers the segment to its socket identified by port 46428. Note that Host B could be running multiple processes, each with its own UDP socket and associated port number. As UDP segments arrive from the network, Host B directs (demultiplexes) each segment to the appropriate socket by examining the segment's destination port number.

It is important to note that a UDP socket is fully identified by a two-tuple consisting of a destination IP address and a destination port number. As a consequence, if two UDP segments have different source IP addresses and/or source port numbers, but have the same *destination* IP address and *destination* port number, then the two segments will be directed to the same destination process via the same destination socket.

You may be wondering now, what is the purpose of the source port number? As shown in Figure 3.4, in the A-to-B segment the source port number serves as part of a “return address”—when B wants to send a segment back to A, the destination port in the B-to-A segment will take its value from the source port value of the A-to-B segment. (The complete return address is A’s IP address and the source port number.) As an example, recall the UDP server program studied in Section 2.7. In `UDPServer.py`, the server uses the `recvfrom()` method to extract the client-side (source) port number from the segment it receives from the client; it then sends a new segment to the client, with the extracted source port number serving as the destination port number in this new segment.

Connection-Oriented Multiplexing and Demultiplexing

In order to understand TCP demultiplexing, we have to take a close look at TCP sockets and TCP connection establishment. One subtle difference between a TCP socket and a UDP socket is that a TCP socket is identified by a four-tuple: (source IP address, source port number, destination IP address, destination port number). Thus, when a TCP segment arrives from the network to a host, the host uses all four values to direct (demultiplex) the segment to the appropriate socket. In particular, and in contrast with UDP, two arriving TCP segments with different source IP

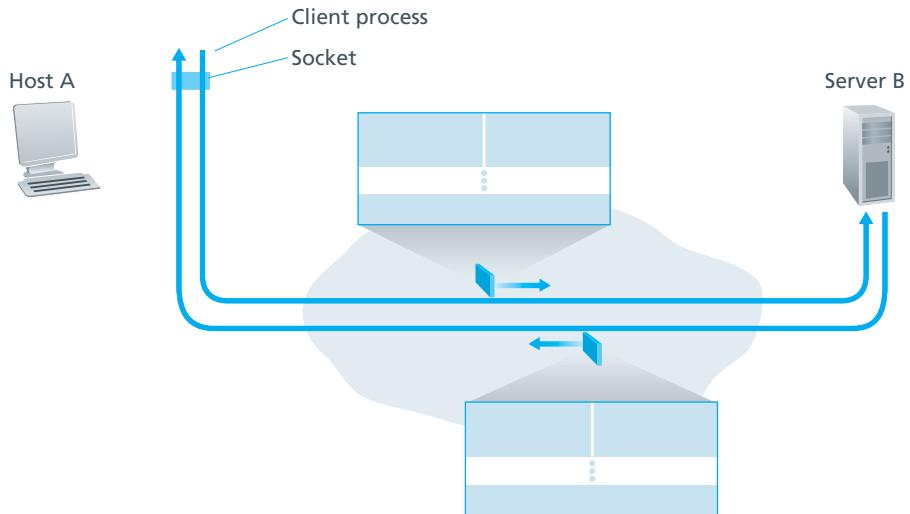


Figure 3.4 ♦ The inversion of source and destination port numbers

addresses or source port numbers will (with the exception of a TCP segment carrying the original connection-establishment request) be directed to two different sockets. To gain further insight, let's reconsider the TCP client-server programming example in Section 2.7.2:

- The TCP server application has a “welcoming socket,” that waits for connection-establishment requests from TCP clients (see Figure 2.29) on port number 12000.
- The TCP client creates a socket and sends a connection establishment request segment with the lines:

```
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, 12000))
```

- A connection-establishment request is nothing more than a TCP segment with destination port number 12000 and a special connection-establishment bit set in the TCP header (discussed in Section 3.5). The segment also includes a source port number that was chosen by the client.
- When the host operating system of the computer running the server process receives the incoming connection-request segment with destination port 12000, it locates the server process that is waiting to accept a connection on port number 12000. The server process then creates a new socket:

```
connectionSocket, addr = serverSocket.accept()
```

- Also, the transport layer at the server notes the following four values in the connection-request segment: (1) the source port number in the segment, (2) the IP address of the source host, (3) the destination port number in the segment, and (4) its own IP address. The newly created connection socket is identified by these four values; all subsequently arriving segments whose source port, source IP address, destination port, and destination IP address match these four values will be demultiplexed to this socket. With the TCP connection now in place, the client and server can now send data to each other.

The server host may support many simultaneous TCP connection sockets, with each socket attached to a process, and with each socket identified by its own four-tuple. When a TCP segment arrives at the host, all four fields (source IP address, source port, destination IP address, destination port) are used to direct (demultiplex) the segment to the appropriate socket.



FOCUS ON SECURITY

PORT SCANNING

We've seen that a server process waits patiently on an open port for contact by a remote client. Some ports are reserved for well-known applications (e.g., Web, FTP, DNS, and SMTP servers); other ports are used by convention by popular applications (e.g., the Microsoft 2000 SQL server listens for requests on UDP port 1434). Thus, if we determine that a port is open on a host, we may be able to map that port to a specific application running on the host. This is very useful for system administrators, who are often interested in knowing which network applications are running on the hosts in their networks. But attackers, in order to "case the joint," also want to know which ports are open on target hosts. If a host is found to be running an application with a known security flaw (e.g., a SQL server listening on port 1434 was subject to a buffer overflow, allowing a remote user to execute arbitrary code on the vulnerable host, a flaw exploited by the Slammer worm [CERT 2003–04]), then that host is ripe for attack.

Determining which applications are listening on which ports is a relatively easy task. Indeed there are a number of public domain programs, called port scanners, that do just that. Perhaps the most widely used of these is nmap, freely available at <http://nmap.org> and included in most Linux distributions. For TCP, nmap sequentially scans ports, looking for ports that are accepting TCP connections. For UDP, nmap again sequentially scans ports, looking for UDP ports that respond to transmitted UDP segments. In both cases, nmap returns a list of open, closed, or unreachable ports. A host running nmap can attempt to scan any target host *anywhere* in the Internet. We'll revisit nmap in Section 3.5.6, when we discuss TCP connection management.

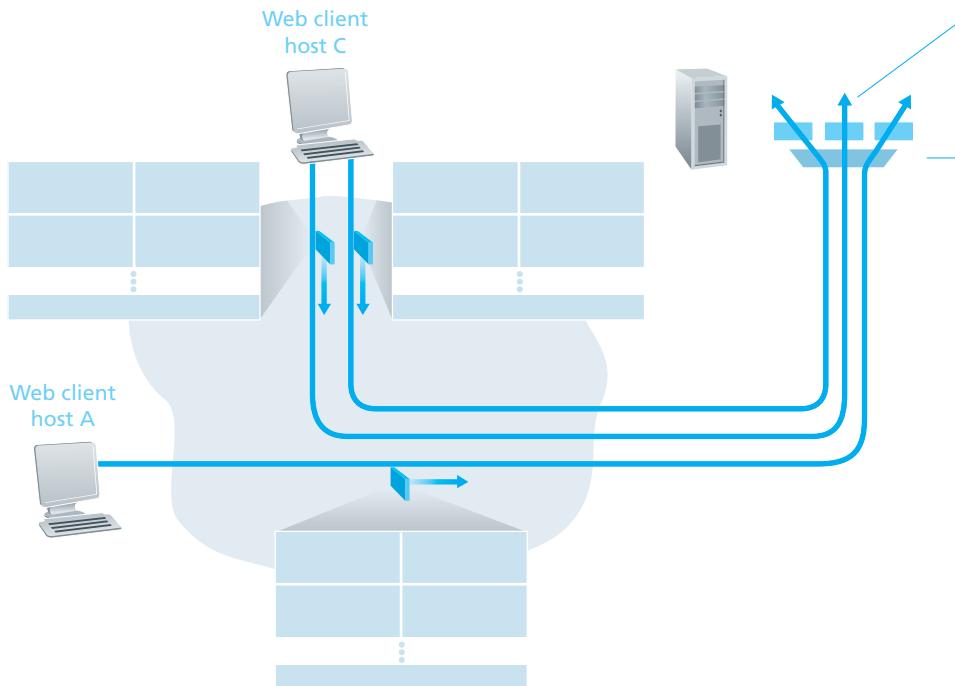


Figure 3.5 ♦ Two clients, using the same destination port number (80) to communicate with the same Web server application

The situation is illustrated in Figure 3.5, in which Host C initiates two HTTP sessions to server B, and Host A initiates one HTTP session to B. Hosts A and C and server B each have their own unique IP address—A, C, and B, respectively. Host C assigns two different source port numbers (26145 and 7532) to its two HTTP connections. Because Host A is choosing source port numbers independently of C, it might also assign a source port of 26145 to its HTTP connection. But this is not a problem—server B will still be able to correctly demultiplex the two connections having the same source port number, since the two connections have different source IP addresses.

Web Servers and TCP

Before closing this discussion, it's instructive to say a few additional words about Web servers and how they use port numbers. Consider a host running a Web server, such as an Apache Web server, on port 80. When clients (for example, browsers) send segments to the server, *all* segments will have destination port 80. In particular, both the initial connection-establishment segments and the segments carrying HTTP request messages will have destination port 80. As we have just described,

the server distinguishes the segments from the different clients using source IP addresses and source port numbers.

Figure 3.5 shows a Web server that spawns a new process for each connection. As shown in Figure 3.5, each of these processes has its own connection socket through which HTTP requests arrive and HTTP responses are sent. We mention, however, that there is not always a one-to-one correspondence between connection sockets and processes. In fact, today's high-performing Web servers often use only one process, and create a new thread with a new connection socket for each new client connection. (A thread can be viewed as a lightweight subprocess.) If you did the first programming assignment in Chapter 2, you built a Web server that does just this. For such a server, at any given time there may be many connection sockets (with different identifiers) attached to the same process.

If the client and server are using persistent HTTP, then throughout the duration of the persistent connection the client and server exchange HTTP messages via the same server socket. However, if the client and server use non-persistent HTTP, then a new TCP connection is created and closed for every request/response, and hence a new socket is created and later closed for every request/response. This frequent creating and closing of sockets can severely impact the performance of a busy Web server (although a number of operating system tricks can be used to mitigate the problem). Readers interested in the operating system issues surrounding persistent and non-persistent HTTP are encouraged to see [Nielsen 1997; Nahum 2002].

Now that we've discussed transport-layer multiplexing and demultiplexing, let's move on and discuss one of the Internet's transport protocols, UDP. In the next section we'll see that UDP adds little more to the network-layer protocol than a multiplexing/demultiplexing service.

3.3 Connectionless Transport: UDP

In this section, we'll take a close look at UDP, how it works, and what it does. We encourage you to refer back to Section 2.1, which includes an overview of the UDP service model, and to Section 2.7.1, which discusses socket programming using UDP.

To motivate our discussion about UDP, suppose you were interested in designing a no-frills, bare-bones transport protocol. How might you go about doing this? You might first consider using a vacuous transport protocol. In particular, on the sending side, you might consider taking the messages from the application process and passing them directly to the network layer; and on the receiving side, you might consider taking the messages arriving from the network layer and passing them directly to the application process. But as we learned in the previous section, we have to do a little more than nothing! At the very least, the transport layer has to

provide a multiplexing/demultiplexing service in order to pass data between the network layer and the correct application-level process.

UDP, defined in [RFC 768], does just about as little as a transport protocol can do. Aside from the multiplexing/demultiplexing function and some light error checking, it adds nothing to IP. In fact, if the application developer chooses UDP instead of TCP, then the application is almost directly talking with IP. UDP takes messages from the application process, attaches source and destination port number fields for the multiplexing/demultiplexing service, adds two other small fields, and passes the resulting segment to the network layer. The network layer encapsulates the transport-layer segment into an IP datagram and then makes a best-effort attempt to deliver the segment to the receiving host. If the segment arrives at the receiving host, UDP uses the destination port number to deliver the segment's data to the correct application process. Note that with UDP there is no handshaking between sending and receiving transport-layer entities before sending a segment. For this reason, UDP is said to be *connectionless*.

DNS is an example of an application-layer protocol that typically uses UDP. When the DNS application in a host wants to make a query, it constructs a DNS query message and passes the message to UDP. Without performing any handshaking with the UDP entity running on the destination end system, the host-side UDP adds header fields to the message and passes the resulting segment to the network layer. The network layer encapsulates the UDP segment into a datagram and sends the datagram to a name server. The DNS application at the querying host then waits for a reply to its query. If it doesn't receive a reply (possibly because the underlying network lost the query or the reply), either it tries sending the query to another name server, or it informs the invoking application that it can't get a reply.

Now you might be wondering why an application developer would ever choose to build an application over UDP rather than over TCP. Isn't TCP always preferable, since TCP provides a reliable data transfer service, while UDP does not? The answer is no, as many applications are better suited for UDP for the following reasons:

- *Finer application-level control over what data is sent, and when.* Under UDP, as soon as an application process passes data to UDP, UDP will package the data inside a UDP segment and immediately pass the segment to the network layer. TCP, on the other hand, has a congestion-control mechanism that throttles the transport-layer TCP sender when one or more links between the source and destination hosts become excessively congested. TCP will also continue to resend a segment until the receipt of the segment has been acknowledged by the destination, regardless of how long reliable delivery takes. Since real-time applications often require a minimum sending rate, do not want to overly delay segment transmission, and can tolerate some data loss, TCP's service model is not particularly well matched to these applications' needs. As discussed below, these applications can use UDP and implement, as part of the application, any additional functionality that is needed beyond UDP's no-frills segment-delivery service.

- *No connection establishment.* As we'll discuss later, TCP uses a three-way handshake before it starts to transfer data. UDP just blasts away without any formal preliminaries. Thus UDP does not introduce any delay to establish a connection. This is probably the principal reason why DNS runs over UDP rather than TCP—DNS would be much slower if it ran over TCP. HTTP uses TCP rather than UDP, since reliability is critical for Web pages with text. But, as we briefly discussed in Section 2.2, the TCP connection-establishment delay in HTTP is an important contributor to the delays associated with downloading Web documents.
- *No connection state.* TCP maintains connection state in the end systems. This connection state includes receive and send buffers, congestion-control parameters, and sequence and acknowledgment number parameters. We will see in Section 3.5 that this state information is needed to implement TCP's reliable data transfer service and to provide congestion control. UDP, on the other hand, does not maintain connection state and does not track any of these parameters. For this reason, a server devoted to a particular application can typically support many more active clients when the application runs over UDP rather than TCP.
- *Small packet header overhead.* The TCP segment has 20 bytes of header overhead in every segment, whereas UDP has only 8 bytes of overhead.

Figure 3.6 lists popular Internet applications and the transport protocols that they use. As we expect, e-mail, remote terminal access, the Web, and file transfer run over TCP—all these applications need the reliable data transfer service of TCP. Nevertheless, many important applications run over UDP rather than TCP. UDP is used for RIP routing table updates (see Section 4.6.1). Since RIP updates are sent periodically (typically every five minutes), lost updates will be replaced by more recent updates, thus making the lost, out-of-date update useless. UDP is also used to carry network management (SNMP; see Chapter 9) data. UDP is preferred to TCP in this case, since network management applications must often run when the network is in a stressed state—precisely when reliable, congestion-controlled data transfer is difficult to achieve. Also, as we mentioned earlier, DNS runs over UDP, thereby avoiding TCP's connection-establishment delays.

As shown in Figure 3.6, both UDP and TCP are used today with multimedia applications, such as Internet phone, real-time video conferencing, and streaming of stored audio and video. We'll take a close look at these applications in Chapter 7. We just mention now that all of these applications can tolerate a small amount of packet loss, so that reliable data transfer is not absolutely critical for the application's success. Furthermore, real-time applications, like Internet phone and video conferencing, react very poorly to TCP's congestion control. For these reasons, developers of multimedia applications may choose to run their applications over UDP instead of TCP. However, TCP is increasingly being used for streaming media transport. For example, [Sripanidkulchai 2004] found that nearly 75% of on-demand and live streaming used TCP. When packet loss rates are low, and with some organizations

Application	Application-Layer Protocol	Underlying Transport Protocol
Electronic mail	SMTP	TCP
Remote terminal access	Telnet	TCP
Web	HTTP	TCP
File transfer	FTP	TCP
Remote file server	NFS	Typically UDP
Streaming multimedia	typically proprietary	UDP or TCP
Internet telephony	typically proprietary	UDP or TCP
Network management	SNMP	Typically UDP
Routing protocol	RIP	Typically UDP
Name translation	DNS	Typically UDP

Figure 3.6 ♦ Popular Internet applications and their underlying transport protocols

blocking UDP traffic for security reasons (see Chapter 8), TCP becomes an increasingly attractive protocol for streaming media transport.

Although commonly done today, running multimedia applications over UDP is controversial. As we mentioned above, UDP has no congestion control. But congestion control is needed to prevent the network from entering a congested state in which very little useful work is done. If everyone were to start streaming high-bit-rate video without using any congestion control, there would be so much packet overflow at routers that very few UDP packets would successfully traverse the source-to-destination path. Moreover, the high loss rates induced by the uncontrolled UDP senders would cause the TCP senders (which, as we'll see, *do* decrease their sending rates in the face of congestion) to dramatically decrease their rates. Thus, the lack of congestion control in UDP can result in high loss rates between a UDP sender and receiver, and the crowding out of TCP sessions—a potentially serious problem [Floyd 1999]. Many researchers have proposed new mechanisms to force all sources, including UDP sources, to perform adaptive congestion control [Mahdavi 1997; Floyd 2000; Kohler 2006: RFC 4340].

Before discussing the UDP segment structure, we mention that it *is* possible for an application to have reliable data transfer when using UDP. This can be done if reliability is built into the application itself (for example, by adding acknowledgment and retransmission mechanisms, such as those we'll study in the next section). But this is a nontrivial task that would keep an application developer busy debugging for

a long time. Nevertheless, building reliability directly into the application allows the application to “have its cake and eat it too.” That is, application processes can communicate reliably without being subjected to the transmission-rate constraints imposed by TCP’s congestion-control mechanism.

3.3.1 UDP Segment Structure

The UDP segment structure, shown in Figure 3.7, is defined in RFC 768. The application data occupies the data field of the UDP segment. For example, for DNS, the data field contains either a query message or a response message. For a streaming audio application, audio samples fill the data field. The UDP header has only four fields, each consisting of two bytes. As discussed in the previous section, the port numbers allow the destination host to pass the application data to the correct process running on the destination end system (that is, to perform the demultiplexing function). The length field specifies the number of bytes in the UDP segment (header plus data). An explicit length value is needed since the size of the data field may differ from one UDP segment to the next. The checksum is used by the receiving host to check whether errors have been introduced into the segment. In truth, the checksum is also calculated over a few of the fields in the IP header in addition to the UDP segment. But we ignore this detail in order to see the forest through the trees. We’ll discuss the checksum calculation below. Basic principles of error detection are described in Section 5.2. The length field specifies the length of the UDP segment, including the header, in bytes.

3.3.2 UDP Checksum

The UDP checksum provides for error detection. That is, the checksum is used to determine whether bits within the UDP segment have been altered (for example, by noise in the links or while stored in a router) as it moved from source to destination. UDP at the sender side performs the 1s complement of the sum of all the 16-bit words in the segment, with any overflow encountered during the sum being

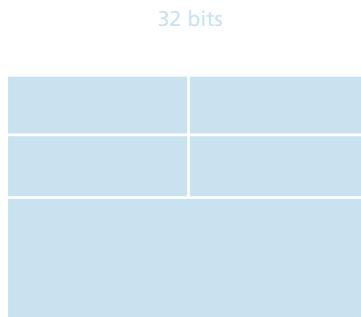


Figure 3.7 ♦ UDP segment structure

wrapped around. This result is put in the checksum field of the UDP segment. Here we give a simple example of the checksum calculation. You can find details about efficient implementation of the calculation in RFC 1071 and performance over real data in [Stone 1998; Stone 2000]. As an example, suppose that we have the following three 16-bit words:

```
0110011001100000  
0101010101010101  
1000111100001100
```

The sum of first two of these 16-bit words is

```
0110011001100000  
0101010101010101  
1011101110110101
```

Adding the third word to the above sum gives

```
1011101110110101  
1000111100001100  
0100101011000010
```

Note that this last addition had overflow, which was wrapped around. The 1s complement is obtained by converting all the 0s to 1s and converting all the 1s to 0s. Thus the 1s complement of the sum 0100101011000010 is 1011010100111101, which becomes the checksum. At the receiver, all four 16-bit words are added, including the checksum. If no errors are introduced into the packet, then clearly the sum at the receiver will be 1111111111111111. If one of the bits is a 0, then we know that errors have been introduced into the packet.

You may wonder why UDP provides a checksum in the first place, as many link-layer protocols (including the popular Ethernet protocol) also provide error checking. The reason is that there is no guarantee that all the links between source and destination provide error checking; that is, one of the links may use a link-layer protocol that does not provide error checking. Furthermore, even if segments are correctly transferred across a link, it's possible that bit errors could be introduced when a segment is stored in a router's memory. Given that neither link-by-link reliability nor in-memory error detection is guaranteed, UDP must provide error detection at the transport layer, *on an end-end basis*, if the end-end data transfer service is to provide error detection. This is an example of the celebrated **end-end principle** in system design [Saltzer 1984], which states that since certain functionality (error detection, in this case) must be implemented on an end-end basis: “functions placed at the lower levels may be redundant or of little value when compared to the cost of providing them at the higher level.”

Because IP is supposed to run over just about any layer-2 protocol, it is useful for the transport layer to provide error checking as a safety measure. Although UDP

provides error checking, it does not do anything to recover from an error. Some implementations of UDP simply discard the damaged segment; others pass the damaged segment to the application with a warning.

That wraps up our discussion of UDP. We will soon see that TCP offers reliable data transfer to its applications as well as other services that UDP doesn't offer. Naturally, TCP is also more complex than UDP. Before discussing TCP, however, it will be useful to step back and first discuss the underlying principles of reliable data transfer.

3.4 Principles of Reliable Data Transfer

In this section, we consider the problem of reliable data transfer in a general context. This is appropriate since the problem of implementing reliable data transfer occurs not only at the transport layer, but also at the link layer and the application layer as well. The general problem is thus of central importance to networking. Indeed, if one had to identify a “top-ten” list of fundamentally important problems in all of networking, this would be a candidate to lead the list. In the next section we'll examine TCP and show, in particular, that TCP exploits many of the principles that we are about to describe.

Figure 3.8 illustrates the framework for our study of reliable data transfer. The service abstraction provided to the upper-layer entities is that of a reliable channel through which data can be transferred. With a reliable channel, no transferred data bits are corrupted (flipped from 0 to 1, or vice versa) or lost, and all are delivered in the order in which they were sent. This is precisely the service model offered by TCP to the Internet applications that invoke it.

It is the responsibility of a **reliable data transfer protocol** to implement this service abstraction. This task is made difficult by the fact that the layer *below* the reliable data transfer protocol may be unreliable. For example, TCP is a reliable data transfer protocol that is implemented on top of an unreliable (IP) end-to-end network layer. More generally, the layer beneath the two reliably communicating end points might consist of a single physical link (as in the case of a link-level data transfer protocol) or a global internetwork (as in the case of a transport-level protocol). For our purposes, however, we can view this lower layer simply as an unreliable point-to-point channel.

In this section, we will incrementally develop the sender and receiver sides of a reliable data transfer protocol, considering increasingly complex models of the underlying channel. For example, we'll consider what protocol mechanisms are needed when the underlying channel can corrupt bits or lose entire packets. One assumption we'll adopt throughout our discussion here is that packets will be delivered in the order in which they were sent, with some packets possibly being lost; that is, the underlying channel will not reorder packets. Figure 3.8(b) illustrates the interfaces for our data transfer protocol. The sending side of the data transfer protocol will be invoked from above by a call to `rdt_send()`. It will pass the data to be delivered to the upper layer at the receiving side. (Here `rdt` stands for *reliable data transfer* protocol and `_send`

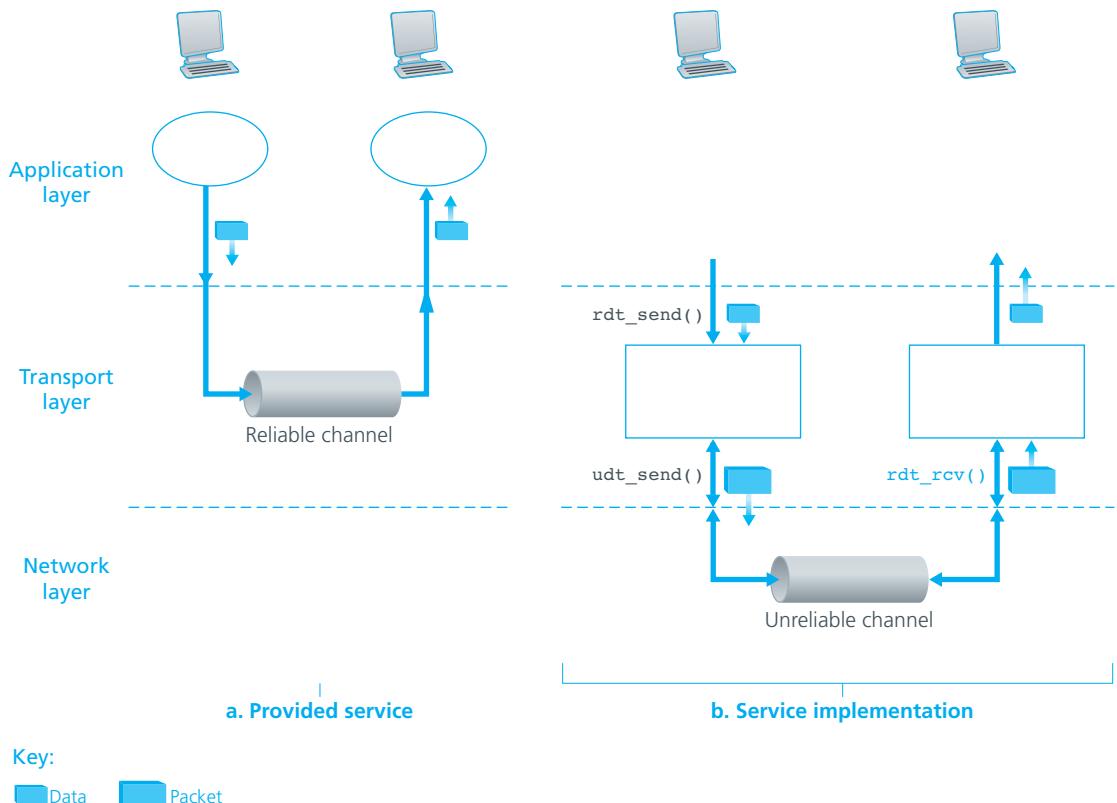


Figure 3.8 ♦ Reliable data transfer: Service model and service implementation

indicates that the sending side of `rdt` is being called. The first step in developing any protocol is to choose a good name!) On the receiving side, `rdt_rcv()` will be called when a packet arrives from the receiving side of the channel. When the `rdt` protocol wants to deliver data to the upper layer, it will do so by calling `deliver_data()`. In the following we use the terminology “packet” rather than transport-layer “segment.” Because the theory developed in this section applies to computer networks in general and not just to the Internet transport layer, the generic term “packet” is perhaps more appropriate here.

In this section we consider only the case of **unidirectional data transfer**, that is, data transfer from the sending to the receiving side. The case of reliable **bidirectional** (that is, full-duplex) **data transfer** is conceptually no more difficult but considerably more tedious to explain. Although we consider only unidirectional data transfer, it is important to note that the sending and receiving sides of our protocol will nonetheless need to transmit packets in *both* directions, as indicated in Figure 3.8. We will see shortly that, in addition to exchanging packets containing the data to be transferred, the

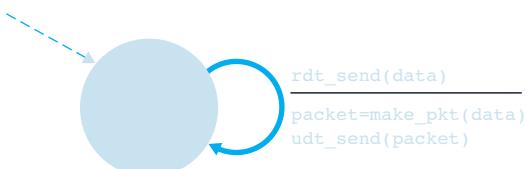
sending and receiving sides of `rdt` will also need to exchange control packets back and forth. Both the send and receive sides of `rdt` send packets to the other side by a call to `udt_send()` (where `udt` stands for *unreliable data transfer*).

3.4.1 Building a Reliable Data Transfer Protocol

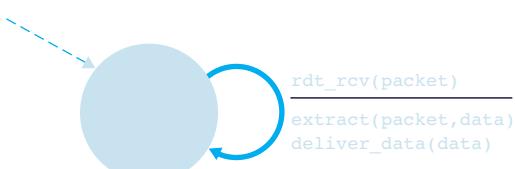
We now step through a series of protocols, each one becoming more complex, arriving at a flawless, reliable data transfer protocol.

Reliable Data Transfer over a Perfectly Reliable Channel: `rdt1.0`

We first consider the simplest case, in which the underlying channel is completely reliable. The protocol itself, which we'll call `rdt1.0`, is trivial. The **finite-state machine (FSM)** definitions for the `rdt1.0` sender and receiver are shown in Figure 3.9. The FSM in Figure 3.9(a) defines the operation of the sender, while the FSM in Figure 3.9(b) defines the operation of the receiver. It is important to note that there are *separate* FSMs for the sender and for the receiver. The sender and receiver FSMs in Figure 3.9 each have just one state. The arrows in the FSM description indicate the transition of the protocol from one state to another. (Since each FSM in Figure 3.9 has just one state, a transition is necessarily from the one state back to itself; we'll see more complicated state diagrams shortly.) The event causing the transition is shown above the horizontal line labeling the transition, and



a. `rdt1.0: sending side`



b. `rdt1.0: receiving side`

Figure 3.9 ♦ `rdt1.0` – A protocol for a completely reliable channel

the actions taken when the event occurs are shown below the horizontal line. When no action is taken on an event, or no event occurs and an action is taken, we'll use the symbol Λ below or above the horizontal, respectively, to explicitly denote the lack of an action or event. The initial state of the FSM is indicated by the dashed arrow. Although the FSMs in Figure 3.9 have but one state, the FSMs we will see shortly have multiple states, so it will be important to identify the initial state of each FSM.

The sending side of `rdt` simply accepts data from the upper layer via the `rdt_send(data)` event, creates a packet containing the data (via the action `make_pkt(data)`) and sends the packet into the channel. In practice, the `rdt_send(data)` event would result from a procedure call (for example, to `rdt_send()`) by the upper-layer application.

On the receiving side, `rdt` receives a packet from the underlying channel via the `rdt_rcv(packet)` event, removes the data from the packet (via the action `extract(packet, data)`) and passes the data up to the upper layer (via the action `deliver_data(data)`). In practice, the `rdt_rcv(packet)` event would result from a procedure call (for example, to `rdt_rcv()`) from the lower-layer protocol.

In this simple protocol, there is no difference between a unit of data and a packet. Also, all packet flow is from the sender to receiver; with a perfectly reliable channel there is no need for the receiver side to provide any feedback to the sender since nothing can go wrong! Note that we have also assumed that the receiver is able to receive data as fast as the sender happens to send data. Thus, there is no need for the receiver to ask the sender to slow down!

Reliable Data Transfer over a Channel with Bit Errors: `rdt2.0`

A more realistic model of the underlying channel is one in which bits in a packet may be corrupted. Such bit errors typically occur in the physical components of a network as a packet is transmitted, propagates, or is buffered. We'll continue to assume for the moment that all transmitted packets are received (although their bits may be corrupted) in the order in which they were sent.

Before developing a protocol for reliably communicating over such a channel, first consider how people might deal with such a situation. Consider how you yourself might dictate a long message over the phone. In a typical scenario, the message taker might say “OK” after each sentence has been heard, understood, and recorded. If the message taker hears a garbled sentence, you’re asked to repeat the garbled sentence. This message-dictation protocol uses both **positive acknowledgments** (“OK”) and **negative acknowledgments** (“Please repeat that.”). These control messages allow the receiver to let the sender know what has been received correctly, and what has been received in error and thus requires repeating. In a computer network setting, reliable data transfer protocols based on such retransmission are known as **ARQ (Automatic Repeat reQuest) protocols**.

Fundamentally, three additional protocol capabilities are required in ARQ protocols to handle the presence of bit errors:

- *Error detection.* First, a mechanism is needed to allow the receiver to detect when bit errors have occurred. Recall from the previous section that UDP uses the Internet checksum field for exactly this purpose. In Chapter 5 we'll examine error-detection and -correction techniques in greater detail; these techniques allow the receiver to detect and possibly correct packet bit errors. For now, we need only know that these techniques require that extra bits (beyond the bits of original data to be transferred) be sent from the sender to the receiver; these bits will be gathered into the packet checksum field of the `rdt2.0` data packet.
- *Receiver feedback.* Since the sender and receiver are typically executing on different end systems, possibly separated by thousands of miles, the only way for the sender to learn of the receiver's view of the world (in this case, whether or not a packet was received correctly) is for the receiver to provide explicit feedback to the sender. The positive (ACK) and negative (NAK) acknowledgment replies in the message-dictation scenario are examples of such feedback. Our `rdt2.0` protocol will similarly send ACK and NAK packets back from the receiver to the sender. In principle, these packets need only be one bit long; for example, a 0 value could indicate a NAK and a value of 1 could indicate an ACK.
- *Retransmission.* A packet that is received in error at the receiver will be retransmitted by the sender.

Figure 3.10 shows the FSM representation of `rdt2.0`, a data transfer protocol employing error detection, positive acknowledgments, and negative acknowledgments.

The send side of `rdt2.0` has two states. In the leftmost state, the send-side protocol is waiting for data to be passed down from the upper layer. When the `rdt_send(data)` event occurs, the sender will create a packet (`sndpkt`) containing the data to be sent, along with a packet checksum (for example, as discussed in Section 3.3.2 for the case of a UDP segment), and then send the packet via the `udt_send(sndpkt)` operation. In the rightmost state, the sender protocol is waiting for an ACK or a NAK packet from the receiver. If an ACK packet is received (the notation `rdt_rcv(rcvpkt) && isACK(rcvpkt)` in Figure 3.10 corresponds to this event), the sender knows that the most recently transmitted packet has been received correctly and thus the protocol returns to the state of waiting for data from the upper layer. If a NAK is received, the protocol retransmits the last packet and waits for an ACK or NAK to be returned by the receiver in response to the retransmitted data packet. It is important to note that when the sender is in the wait-for-ACK-or-NAK state, it *cannot* get more data from the upper layer; that is, the `rdt_send()` event can not occur; that will happen only after the sender receives an ACK and leaves this state. Thus, the sender will not send a new piece of data until it is sure that the receiver has

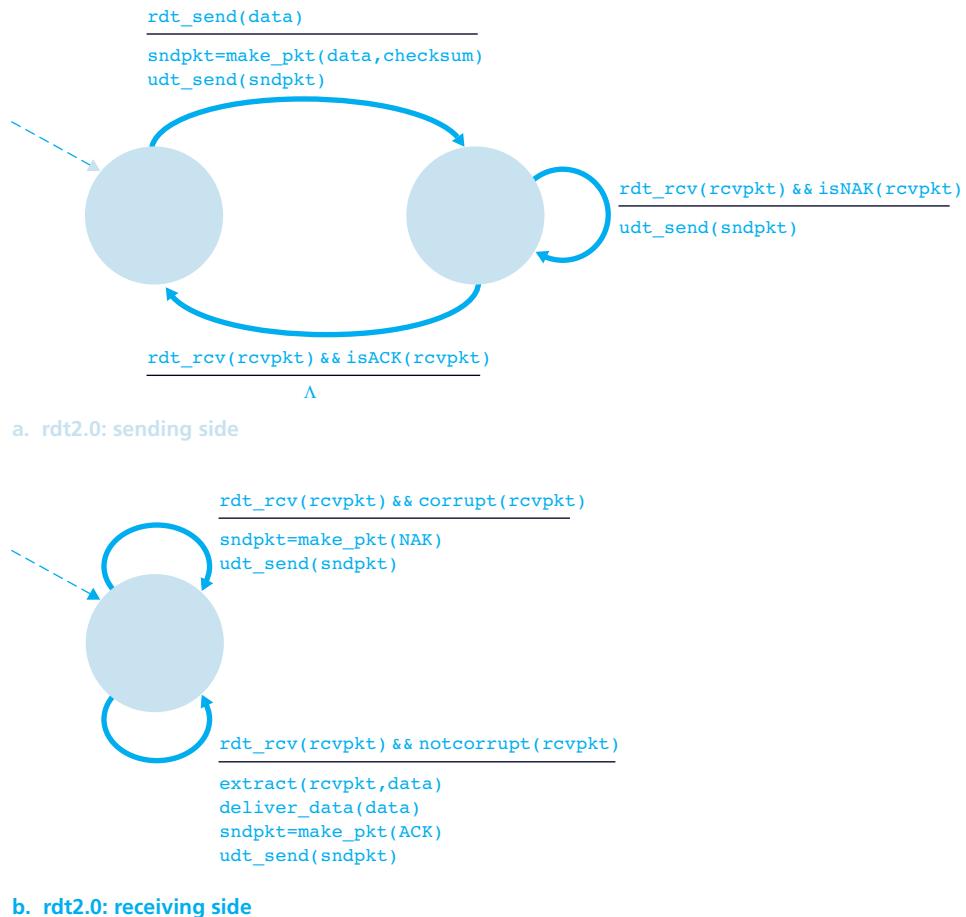


Figure 3.10 ♦ rdt2.0—A protocol for a channel with bit errors

correctly received the current packet. Because of this behavior, protocols such as rdt2.0 are known as **stop-and-wait** protocols.

The receiver-side FSM for rdt2.0 still has a single state. On packet arrival, the receiver replies with either an ACK or a NAK, depending on whether or not the received packet is corrupted. In Figure 3.10, the notation `rdt_rcv(rcvpkt) && corrupt(rcvpkt)` corresponds to the event in which a packet is received and is found to be in error.

Protocol rdt2.0 may look as if it works but, unfortunately, it has a fatal flaw. In particular, we haven't accounted for the possibility that the ACK or NAK packet could be corrupted! (Before proceeding on, you should think about how this

problem may be fixed.) Unfortunately, our slight oversight is not as innocuous as it may seem. Minimally, we will need to add checksum bits to ACK/NAK packets in order to detect such errors. The more difficult question is how the protocol should recover from errors in ACK or NAK packets. The difficulty here is that if an ACK or NAK is corrupted, the sender has no way of knowing whether or not the receiver has correctly received the last piece of transmitted data.

Consider three possibilities for handling corrupted ACKs or NAKs:

- For the first possibility, consider what a human might do in the message-dictation scenario. If the speaker didn't understand the "OK" or "Please repeat that" reply from the receiver, the speaker would probably ask, "What did you say?" (thus introducing a new type of sender-to-receiver packet to our protocol). The receiver would then repeat the reply. But what if the speaker's "What did you say?" is corrupted? The receiver, having no idea whether the garbled sentence was part of the dictation or a request to repeat the last reply, would probably then respond with "What did *you* say?" And then, of course, that response might be garbled. Clearly, we're heading down a difficult path.
- A second alternative is to add enough checksum bits to allow the sender not only to detect, but also to recover from, bit errors. This solves the immediate problem for a channel that can corrupt packets but not lose them.
- A third approach is for the sender simply to resend the current data packet when it receives a garbled ACK or NAK packet. This approach, however, introduces **duplicate packets** into the sender-to-receiver channel. The fundamental difficulty with duplicate packets is that the receiver doesn't know whether the ACK or NAK it last sent was received correctly at the sender. Thus, it cannot know *a priori* whether an arriving packet contains new data or is a retransmission!

A simple solution to this new problem (and one adopted in almost all existing data transfer protocols, including TCP) is to add a new field to the data packet and have the sender number its data packets by putting a **sequence number** into this field. The receiver then need only check this sequence number to determine whether or not the received packet is a retransmission. For this simple case of a stop-and-wait protocol, a 1-bit sequence number will suffice, since it will allow the receiver to know whether the sender is resending the previously transmitted packet (the sequence number of the received packet has the same sequence number as the most recently received packet) or a new packet (the sequence number changes, moving "forward" in modulo-2 arithmetic). Since we are currently assuming a channel that does not lose packets, ACK and NAK packets do not themselves need to indicate the sequence number of the packet they are acknowledging. The sender knows that a received ACK or NAK packet (whether garbled or not) was generated in response to its most recently transmitted data packet.

Figures 3.11 and 3.12 show the FSM description for `rdt2.1`, our fixed version of `rdt2.0`. The `rdt2.1` sender and receiver FSMs each now have twice as many states as before. This is because the protocol state must now reflect whether the packet currently being sent (by the sender) or expected (at the receiver) should have a sequence number of 0 or 1. Note that the actions in those states where a 0-numbered packet is being sent or expected are mirror images of those where a 1-numbered packet is being sent or expected; the only differences have to do with the handling of the sequence number.

Protocol `rdt2.1` uses both positive and negative acknowledgments from the receiver to the sender. When an out-of-order packet is received, the receiver sends a positive acknowledgment for the packet it has received. When a corrupted packet is received, the receiver sends a negative acknowledgment. We can accomplish the same effect as a NAK if, instead of sending a NAK, we send an ACK for the last correctly received packet. A sender that receives two ACKs for the same packet (that is, receives **duplicate ACKs**) knows that the receiver did not correctly receive the packet following the packet that is being ACKed twice. Our NAK-free reliable data

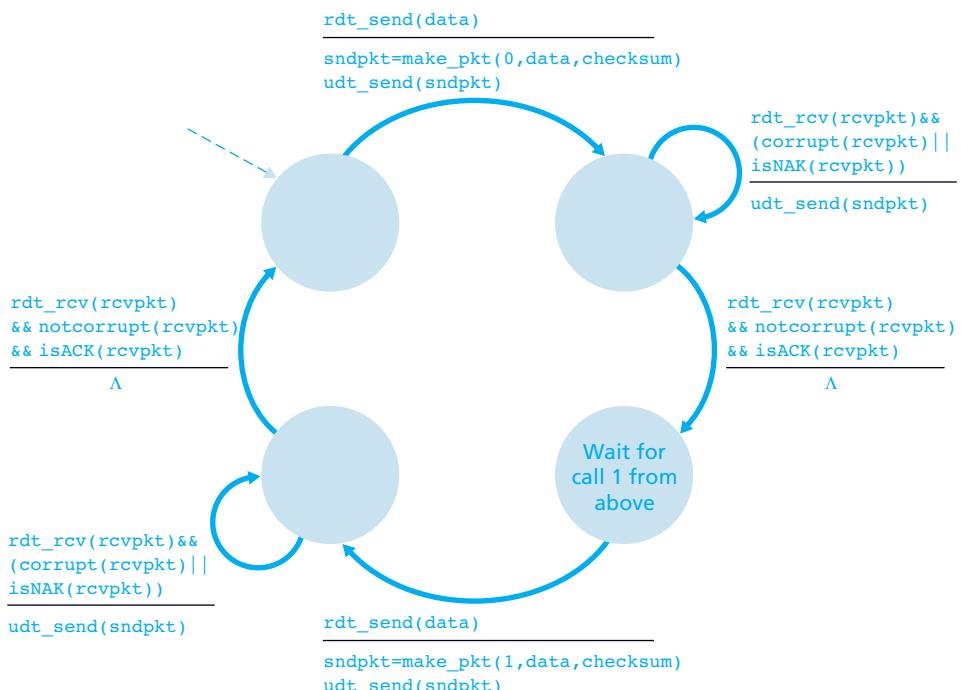


Figure 3.11 ♦ `rdt2.1` sender

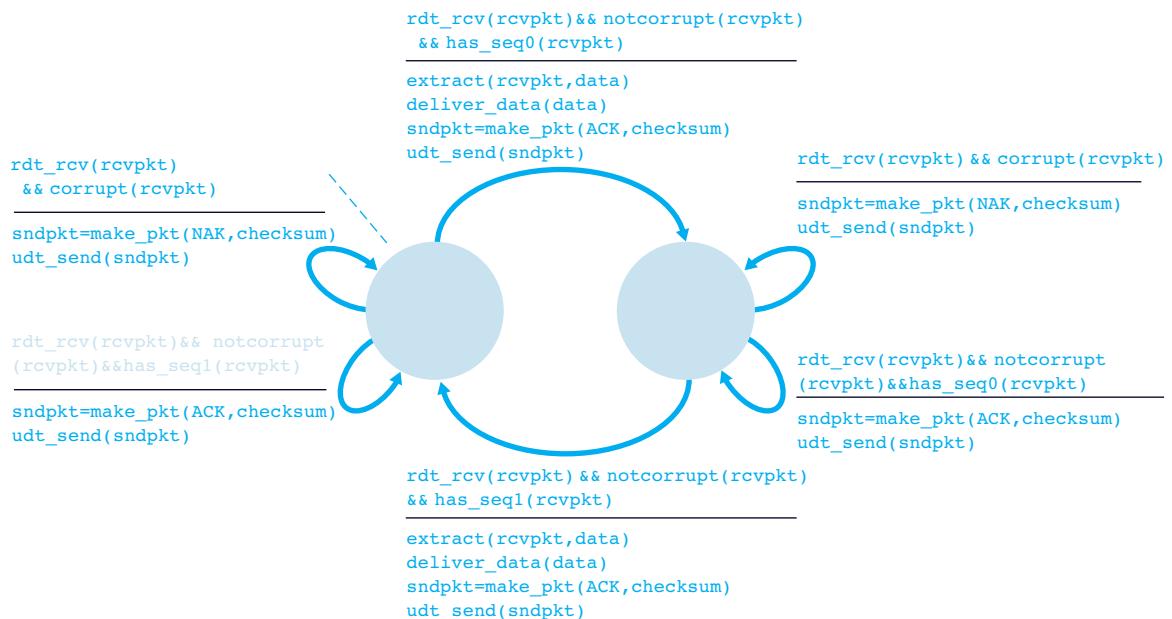


Figure 3.12 ♦ rdt2.1 receiver

transfer protocol for a channel with bit errors is `rdt2.2`, shown in Figures 3.13 and 3.14. One subtle change between `rdt2.1` and `rdt2.2` is that the receiver must now include the sequence number of the packet being acknowledged by an ACK message (this is done by including the `ACK,0` or `ACK,1` argument in `make_pkt()` in the receiver FSM), and the sender must now check the sequence number of the packet being acknowledged by a received ACK message (this is done by including the `0` or `1` argument in `isACK()` in the sender FSM).

Reliable Data Transfer over a Lossy Channel with Bit Errors: `rdt3.0`

Suppose now that in addition to corrupting bits, the underlying channel can *lose* packets as well, a not-uncommon event in today’s computer networks (including the Internet). Two additional concerns must now be addressed by the protocol: how to detect packet loss and what to do when packet loss occurs. The use of checksumming, sequence numbers, ACK packets, and retransmissions—the techniques already developed in `rdt2.2`—will allow us to answer the latter concern. Handling the first concern will require adding a new protocol mechanism.

There are many possible approaches toward dealing with packet loss (several more of which are explored in the exercises at the end of the chapter). Here, we’ll put the burden of detecting and recovering from lost packets on the sender. Suppose

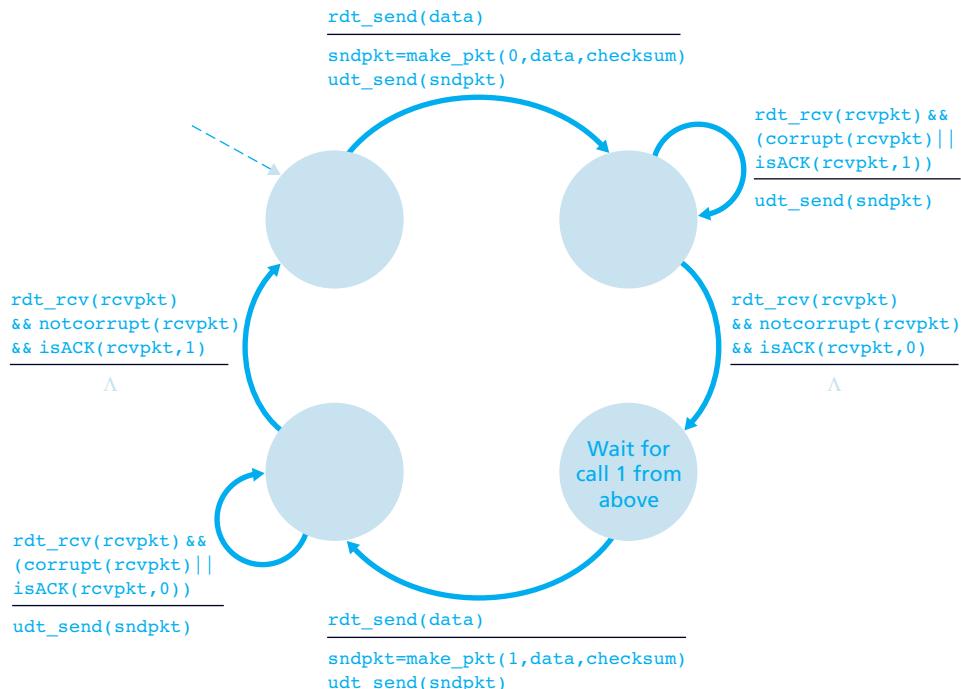


Figure 3.13 ♦ rdt2.2 sender

that the sender transmits a data packet and either that packet, or the receiver's ACK of that packet, gets lost. In either case, no reply is forthcoming at the sender from the receiver. If the sender is willing to wait long enough so that it is *certain* that a packet has been lost, it can simply retransmit the data packet. You should convince yourself that this protocol does indeed work.

But how long must the sender wait to be certain that something has been lost? The sender must clearly wait at least as long as a round-trip delay between the sender and receiver (which may include buffering at intermediate routers) plus whatever amount of time is needed to process a packet at the receiver. In many networks, this worst-case maximum delay is very difficult even to estimate, much less know with certainty. Moreover, the protocol should ideally recover from packet loss as soon as possible; waiting for a worst-case delay could mean a long wait until error recovery is initiated. The approach thus adopted in practice is for the sender to judiciously choose a time value such that packet loss is likely, although not guaranteed, to have happened. If an ACK is not received within this time, the packet is retransmitted. Note that if a packet experiences a particularly large delay, the sender may retransmit the packet even though neither the data packet nor its ACK have been lost. This introduces the possibility of **duplicate data packets** in

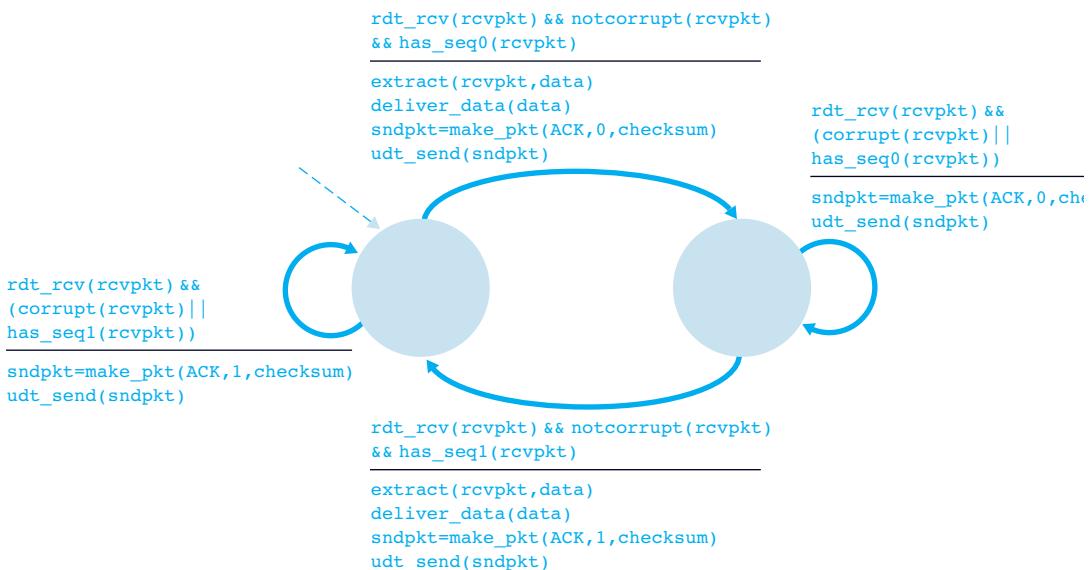


Figure 3.14 ♦ rdt2.2 receiver

the sender-to-receiver channel. Happily, protocol rdt2.2 already has enough functionality (that is, sequence numbers) to handle the case of duplicate packets.

From the sender's viewpoint, retransmission is a panacea. The sender does not know whether a data packet was lost, an ACK was lost, or if the packet or ACK was simply overly delayed. In all cases, the action is the same: retransmit. Implementing a time-based retransmission mechanism requires a **countdown timer** that can interrupt the sender after a given amount of time has expired. The sender will thus need to be able to (1) start the timer each time a packet (either a first-time packet or a retransmission) is sent, (2) respond to a timer interrupt (taking appropriate actions), and (3) stop the timer.

Figure 3.15 shows the sender FSM for rdt3.0, a protocol that reliably transfers data over a channel that can corrupt or lose packets; in the homework problems, you'll be asked to provide the receiver FSM for rdt3.0. Figure 3.16 shows how the protocol operates with no lost or delayed packets and how it handles lost data packets. In Figure 3.16, time moves forward from the top of the diagram toward the bottom of the diagram; note that a receive time for a packet is necessarily later than the send time for a packet as a result of transmission and propagation delays. In Figures 3.16(b)–(d), the send-side brackets indicate the times at which a timer is set and later times out. Several of the more subtle aspects of this protocol are explored in the exercises at the end of this chapter. Because packet sequence numbers alternate between 0 and 1, protocol rdt3.0 is sometimes known as the **alternating-bit protocol**.

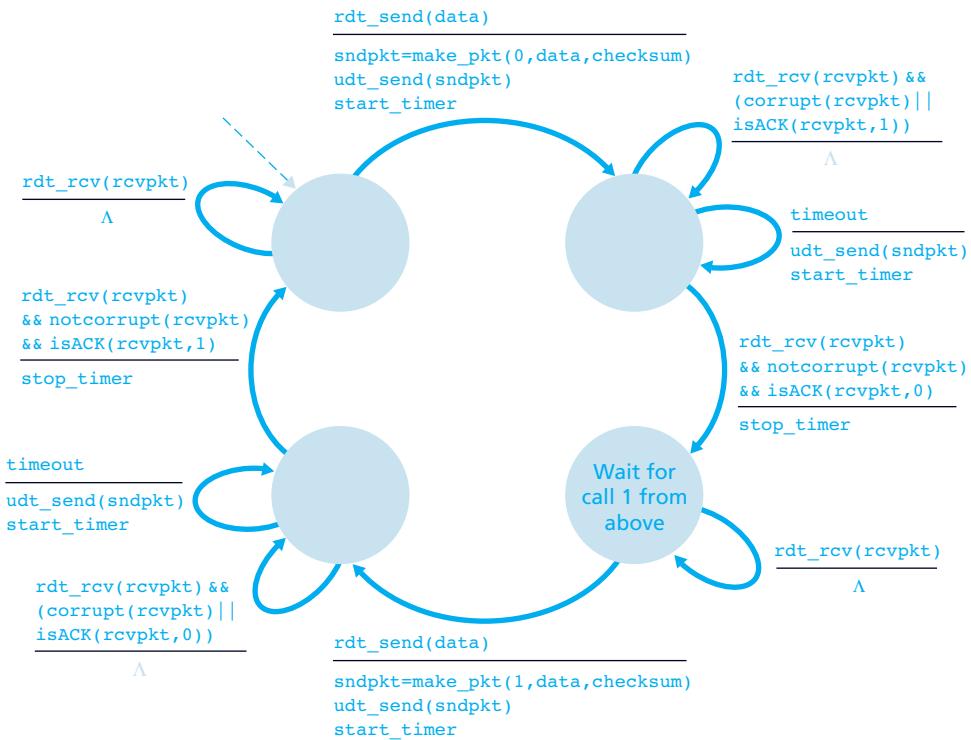


Figure 3.15 ♦ rdt3.0 sender

We have now assembled the key elements of a data transfer protocol. Checksums, sequence numbers, timers, and positive and negative acknowledgment packets each play a crucial and necessary role in the operation of the protocol. We now have a working reliable data transfer protocol!



3.4.2 Pipelined Reliable Data Transfer Protocols

Protocol rdt3.0 is a functionally correct protocol, but it is unlikely that anyone would be happy with its performance, particularly in today's high-speed networks. At the heart of rdt3.0's performance problem is the fact that it is a stop-and-wait protocol.

To appreciate the performance impact of this stop-and-wait behavior, consider an idealized case of two hosts, one located on the West Coast of the United States and the other located on the East Coast, as shown in Figure 3.17. The speed-of-light round-trip propagation delay between these two end systems, RTT, is approximately 30 milliseconds. Suppose that they are connected by a channel with a transmission rate, R , of 1 Gbps (10^9 bits per second). With a packet size, L , of 1,000 bytes

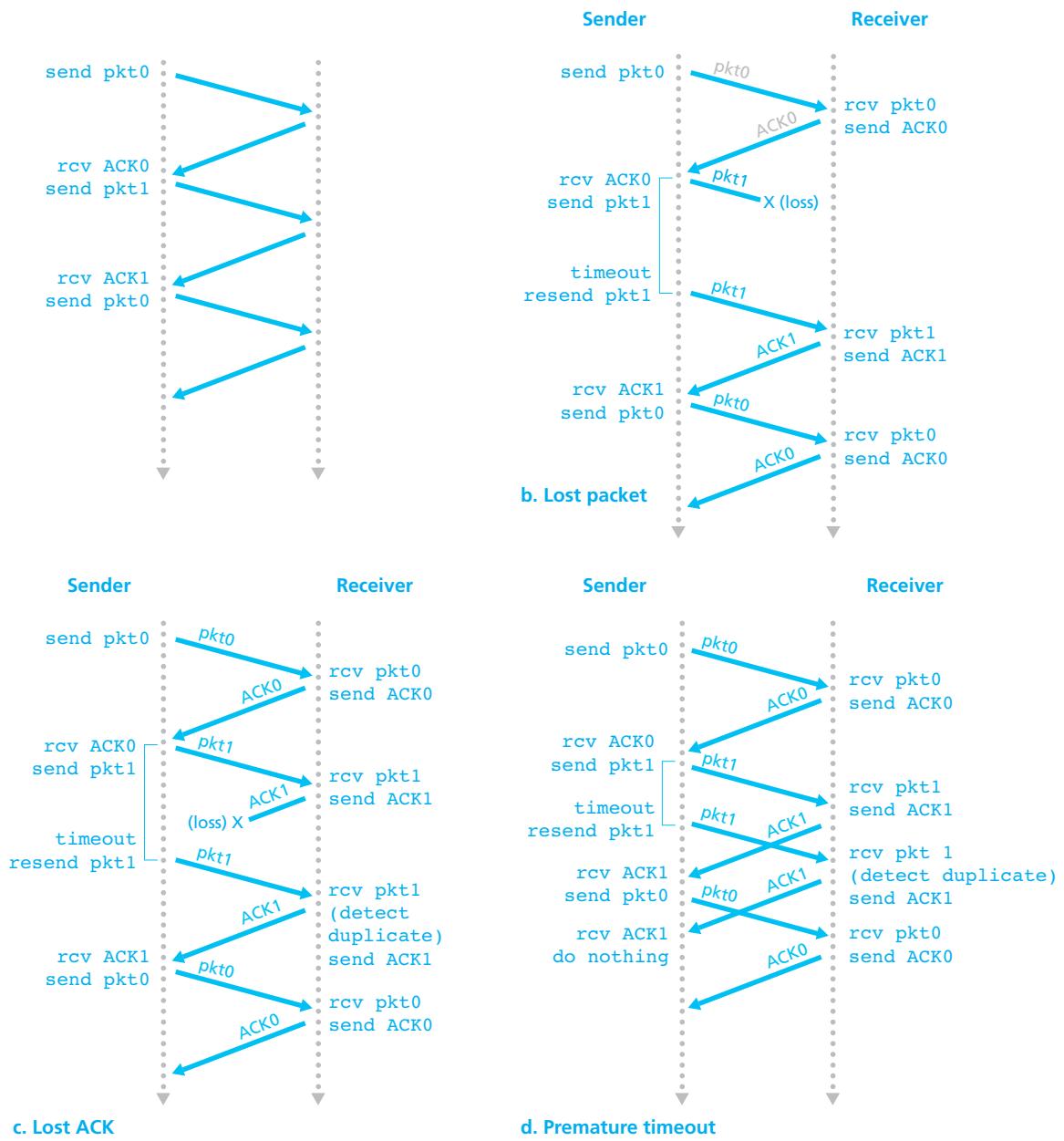


Figure 3.16 ♦ Operation of rdt3.0, the alternating-bit protocol

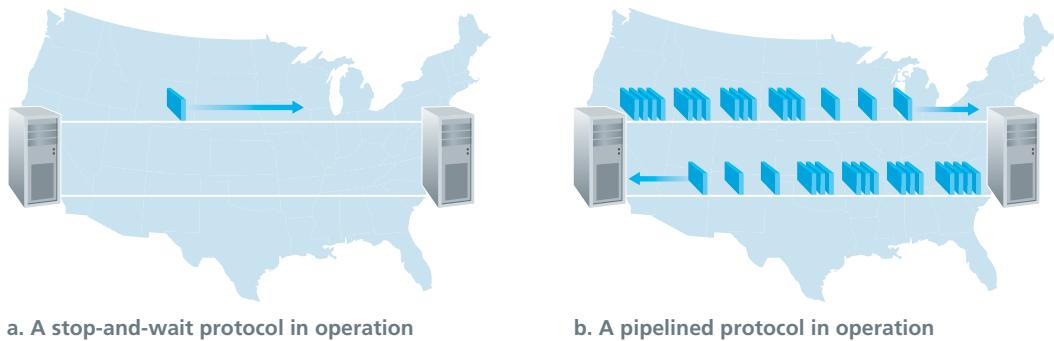


Figure 3.17 ♦ Stop-and-wait versus pipelined protocol

(8,000 bits) per packet, including both header fields and data, the time needed to actually transmit the packet into the 1 Gbps link is

$$d_{trans} = \frac{L}{R} = \frac{8000 \text{ bits}/\text{packet}}{10^9 \text{ bits/sec}} = 8 \text{ microseconds}$$

Figure 3.18(a) shows that with our stop-and-wait protocol, if the sender begins sending the packet at $t = 0$, then at $t = L/R = 8$ microseconds, the last bit enters the channel at the sender side. The packet then makes its 15-msec cross-country journey, with the last bit of the packet emerging at the receiver at $t = RTT/2 + L/R = 15.008$ msec. Assuming for simplicity that ACK packets are extremely small (so that we can ignore their transmission time) and that the receiver can send an ACK as soon as the last bit of a data packet is received, the ACK emerges back at the sender at $t = RTT + L/R = 30.008$ msec. At this point, the sender can now transmit the next message. Thus, in 30.008 msec, the sender was sending for only 0.008 msec. If we define the **utilization** of the sender (or the channel) as the fraction of time the sender is actually busy sending bits into the channel, the analysis in Figure 3.18(a) shows that the stop-and-wait protocol has a rather dismal sender utilization, U_{sender} , of

$$U_{\text{sender}} = \frac{L/R}{RTT + L/R} = \frac{.008}{30.008} = 0.00027$$

That is, the sender was busy only 2.7 hundredths of one percent of the time! Viewed another way, the sender was able to send only 1,000 bytes in 30.008 milliseconds, an effective throughput of only 267 kbps—even though a 1 Gbps link was available! Imagine the unhappy network manager who just paid a fortune for a gigabit capacity link but manages to get a throughput of only 267 kilobits per second! This is a graphic example of how network protocols can limit the capabilities

provided by the underlying network hardware. Also, we have neglected lower-layer protocol-processing times at the sender and receiver, as well as the processing and queuing delays that would occur at any intermediate routers between the sender and receiver. Including these effects would serve only to further increase the delay and further accentuate the poor performance.

The solution to this particular performance problem is simple: Rather than operate in a stop-and-wait manner, the sender is allowed to send multiple packets without waiting for acknowledgments, as illustrated in Figure 3.17(b). Figure 3.18(b) shows that if the sender is allowed to transmit three packets before having to wait for acknowledgments, the utilization of the sender is essentially tripled. Since the many in-transit sender-to-receiver packets can be visualized as filling a pipeline, this technique is known as **pipelining**. Pipelining has the following consequences for reliable data transfer protocols:

- The range of sequence numbers must be increased, since each in-transit packet (not counting retransmissions) must have a unique sequence number and there may be multiple, in-transit, unacknowledged packets.
- The sender and receiver sides of the protocols may have to buffer more than one packet. Minimally, the sender will have to buffer packets that have been transmitted but not yet acknowledged. Buffering of correctly received packets may also be needed at the receiver, as discussed below.
- The range of sequence numbers needed and the buffering requirements will depend on the manner in which a data transfer protocol responds to lost, corrupted, and overly delayed packets. Two basic approaches toward pipelined error recovery can be identified: **Go-Back-N** and **selective repeat**.

3.4.3 Go-Back-N (GBN)

In a **Go-Back-N (GBN) protocol**, the sender is allowed to transmit multiple packets (when available) without waiting for an acknowledgment, but is constrained to have no more than some maximum allowable number, N , of unacknowledged packets in the pipeline. We describe the GBN protocol in some detail in this section. But before reading on, you are encouraged to play with the GBN applet (an awesome applet!) at the companion Web site.

Figure 3.19 shows the sender's view of the range of sequence numbers in a GBN protocol. If we define **base** to be the sequence number of the oldest unacknowledged packet and **nextseqnum** to be the smallest unused sequence number (that is, the sequence number of the next packet to be sent), then four intervals in the range of sequence numbers can be identified. Sequence numbers in the interval $[0, \text{base}-1]$ correspond to packets that have already been transmitted and acknowledged. The interval $[\text{base}, \text{nextseqnum}-1]$ corresponds to packets that have been sent but not yet acknowledged. Sequence numbers in the interval $[\text{nextseqnum}, \text{base}+N-1]$ can

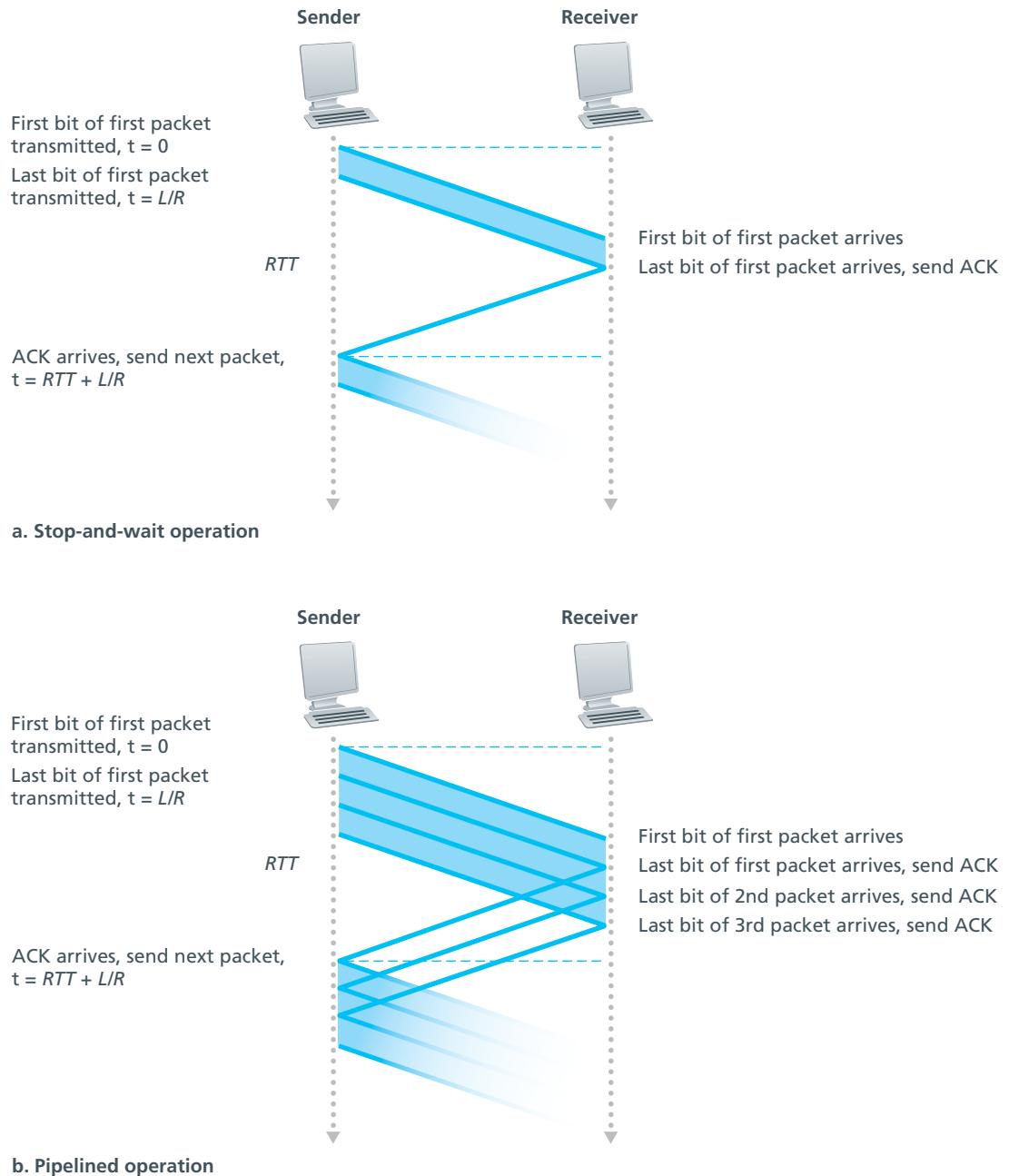


Figure 3.18 ♦ Stop-and-wait and pipelined sending



Figure 3.19 ♦ Sender’s view of sequence numbers in Go-Back-N

be used for packets that can be sent immediately, should data arrive from the upper layer. Finally, sequence numbers greater than or equal to `base+N` cannot be used until an unacknowledged packet currently in the pipeline (specifically, the packet with sequence number `base`) has been acknowledged.

As suggested by Figure 3.19, the range of permissible sequence numbers for transmitted but not yet acknowledged packets can be viewed as a window of size N over the range of sequence numbers. As the protocol operates, this window slides forward over the sequence number space. For this reason, N is often referred to as the **window size** and the GBN protocol itself as a **sliding-window protocol**. You might be wondering why we would even limit the number of outstanding, unacknowledged packets to a value of N in the first place. Why not allow an unlimited number of such packets? We’ll see in Section 3.5 that flow control is one reason to impose a limit on the sender. We’ll examine another reason to do so in Section 3.7, when we study TCP congestion control.

In practice, a packet’s sequence number is carried in a fixed-length field in the packet header. If k is the number of bits in the packet sequence number field, the range of sequence numbers is thus $[0, 2^k - 1]$. With a finite range of sequence numbers, all arithmetic involving sequence numbers must then be done using modulo 2^k arithmetic. (That is, the sequence number space can be thought of as a ring of size 2^k , where sequence number $2^k - 1$ is immediately followed by sequence number 0.) Recall that `rdt3.0` had a 1-bit sequence number and a range of sequence numbers of $[0, 1]$. Several of the problems at the end of this chapter explore the consequences of a finite range of sequence numbers. We will see in Section 3.5 that TCP has a 32-bit sequence number field, where TCP sequence numbers count bytes in the byte stream rather than packets.

Figures 3.20 and 3.21 give an extended FSM description of the sender and receiver sides of an ACK-based, NAK-free, GBN protocol. We refer to this FSM description as an *extended FSM* because we have added variables (similar to programming-language variables) for `base` and `nextseqnum`, and added operations on these variables and conditional actions involving these variables. Note that the extended FSM specification is now beginning to look somewhat like a programming-language specification. [Bochman 1984] provides an excellent survey of additional extensions to FSM techniques as well as other programming-language-based techniques for specifying protocols.

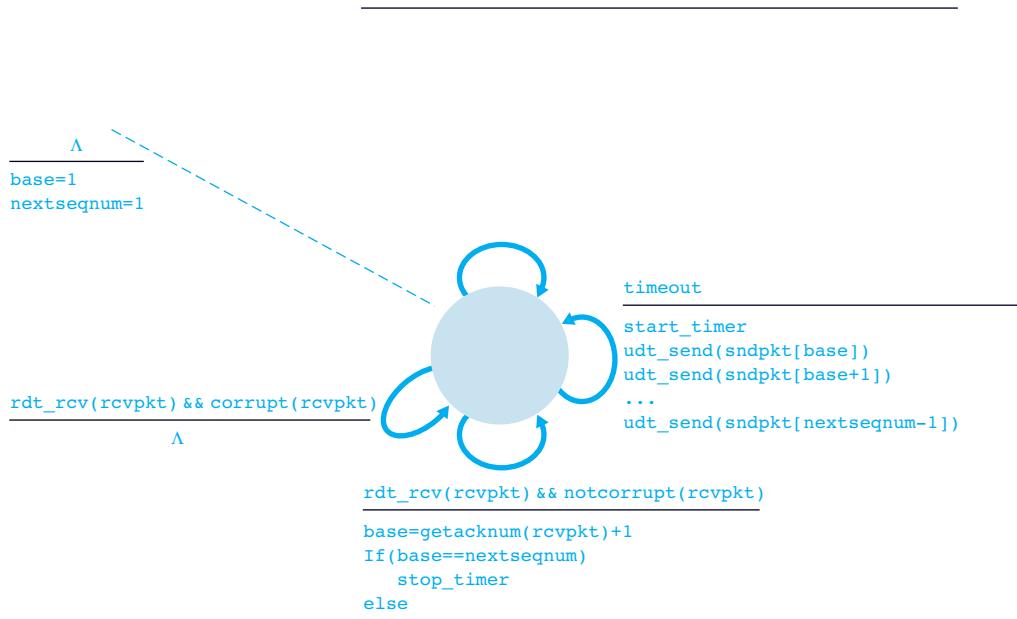


Figure 3.20 ♦ Extended FSM description of GBN sender

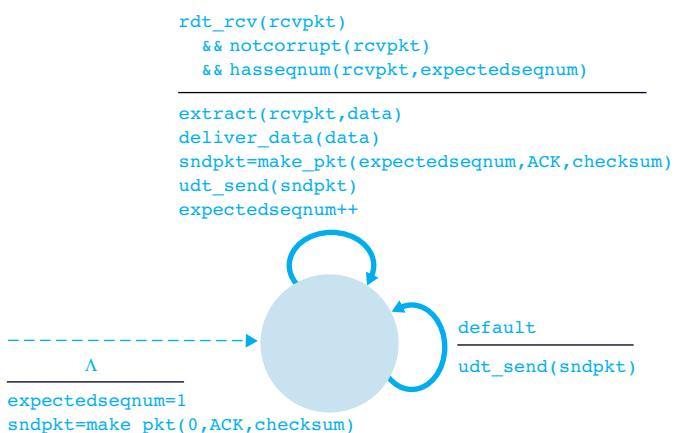


Figure 3.21 ♦ Extended FSM description of GBN receiver

The GBN sender must respond to three types of events:

- *Invocation from above.* When `rdt_send()` is called from above, the sender first checks to see if the window is full, that is, whether there are N outstanding, unacknowledged packets. If the window is not full, a packet is created and sent, and variables are appropriately updated. If the window is full, the sender simply returns the data back to the upper layer, an implicit indication that the window is full. The upper layer would presumably then have to try again later. In a real implementation, the sender would more likely have either buffered (but not immediately sent) this data, or would have a synchronization mechanism (for example, a semaphore or a flag) that would allow the upper layer to call `rdt_send()` only when the window is not full.
- *Receipt of an ACK.* In our GBN protocol, an acknowledgment for a packet with sequence number n will be taken to be a **cumulative acknowledgment**, indicating that all packets with a sequence number up to and including n have been correctly received at the receiver. We'll come back to this issue shortly when we examine the receiver side of GBN.
- *A timeout event.* The protocol's name, "Go-Back-N," is derived from the sender's behavior in the presence of lost or overly delayed packets. As in the stop-and-wait protocol, a timer will again be used to recover from lost data or acknowledgment packets. If a timeout occurs, the sender resends *all* packets that have been previously sent but that have not yet been acknowledged. Our sender in Figure 3.20 uses only a single timer, which can be thought of as a timer for the oldest transmitted but not yet acknowledged packet. If an ACK is received but there are still additional transmitted but not yet acknowledged packets, the timer is restarted. If there are no outstanding, unacknowledged packets, the timer is stopped.

The receiver's actions in GBN are also simple. If a packet with sequence number n is received correctly and is in order (that is, the data last delivered to the upper layer came from a packet with sequence number $n - 1$), the receiver sends an ACK for packet n and delivers the data portion of the packet to the upper layer. In all other cases, the receiver discards the packet and resends an ACK for the most recently received in-order packet. Note that since packets are delivered one at a time to the upper layer, if packet k has been received and delivered, then all packets with a sequence number lower than k have also been delivered. Thus, the use of cumulative acknowledgments is a natural choice for GBN.

In our GBN protocol, the receiver discards out-of-order packets. Although it may seem silly and wasteful to discard a correctly received (but out-of-order) packet, there is some justification for doing so. Recall that the receiver must deliver data in order to the upper layer. Suppose now that packet n is expected, but packet $n + 1$ arrives. Because data must be delivered in order, the receiver *could* buffer (save) packet $n + 1$ and then deliver this packet to the upper layer after it had later

received and delivered packet n . However, if packet n is lost, both it and packet $n + 1$ will eventually be retransmitted as a result of the GBN retransmission rule at the sender. Thus, the receiver can simply discard packet $n + 1$. The advantage of this approach is the simplicity of receiver buffering—the receiver need not buffer *any* out-of-order packets. Thus, while the sender must maintain the upper and lower bounds of its window and the position of `nextseqnum` within this window, the only piece of information the receiver need maintain is the sequence number of the next in-order packet. This value is held in the variable `expectedseqnum`, shown in the receiver FSM in Figure 3.21. Of course, the disadvantage of throwing away a correctly received packet is that the subsequent retransmission of that packet might be lost or garbled and thus even more retransmissions would be required.

Figure 3.22 shows the operation of the GBN protocol for the case of a window size of four packets. Because of this window size limitation, the sender sends packets 0 through 3 but then must wait for one or more of these packets to be acknowledged before proceeding. As each successive ACK (for example, ACK0 and ACK1) is received, the window slides forward and the sender can transmit one new packet (pkt4 and pkt5, respectively). On the receiver side, packet 2 is lost and thus packets 3, 4, and 5 are found to be out of order and are discarded.

Before closing our discussion of GBN, it is worth noting that an implementation of this protocol in a protocol stack would likely have a structure similar to that of the extended FSM in Figure 3.20. The implementation would also likely be in the form of various procedures that implement the actions to be taken in response to the various events that can occur. In such **event-based programming**, the various procedures are called (invoked) either by other procedures in the protocol stack, or as the result of an interrupt. In the sender, these events would be (1) a call from the upper-layer entity to invoke `rdt_send()`, (2) a timer interrupt, and (3) a call from the lower layer to invoke `rdt_rcv()` when a packet arrives. The programming exercises at the end of this chapter will give you a chance to actually implement these routines in a simulated, but realistic, network setting.

We note here that the GBN protocol incorporates almost all of the techniques that we will encounter when we study the reliable data transfer components of TCP in Section 3.5. These techniques include the use of sequence numbers, cumulative acknowledgments, checksums, and a timeout/retransmit operation.

3.4.4 Selective Repeat (SR)

The GBN protocol allows the sender to potentially “fill the pipeline” in Figure 3.17 with packets, thus avoiding the channel utilization problems we noted with stop-and-wait protocols. There are, however, scenarios in which GBN itself suffers from performance problems. In particular, when the window size and bandwidth-delay product are both large, many packets can be in the pipeline. A single packet error can thus cause GBN to retransmit a large number of packets, many unnecessarily. As the probability of channel errors increases, the pipeline can become filled with

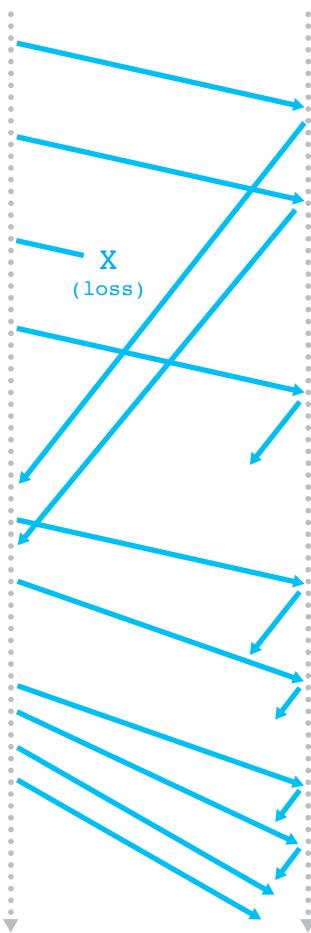


Figure 3.22 ♦ Go-Back-N in operation

these unnecessary retransmissions. Imagine, in our message-dictation scenario, that if every time a word was garbled, the surrounding 1,000 words (for example, a window size of 1,000 words) had to be repeated. The dictation would be slowed by all of the reiterated words.

As the name suggests, selective-repeat protocols avoid unnecessary retransmissions by having the sender retransmit only those packets that it suspects were received in error (that is, were lost or corrupted) at the receiver. This individual, as-needed, retransmission will require that the receiver *individually* acknowledge correctly received packets. A window size of N will again be used to limit the number

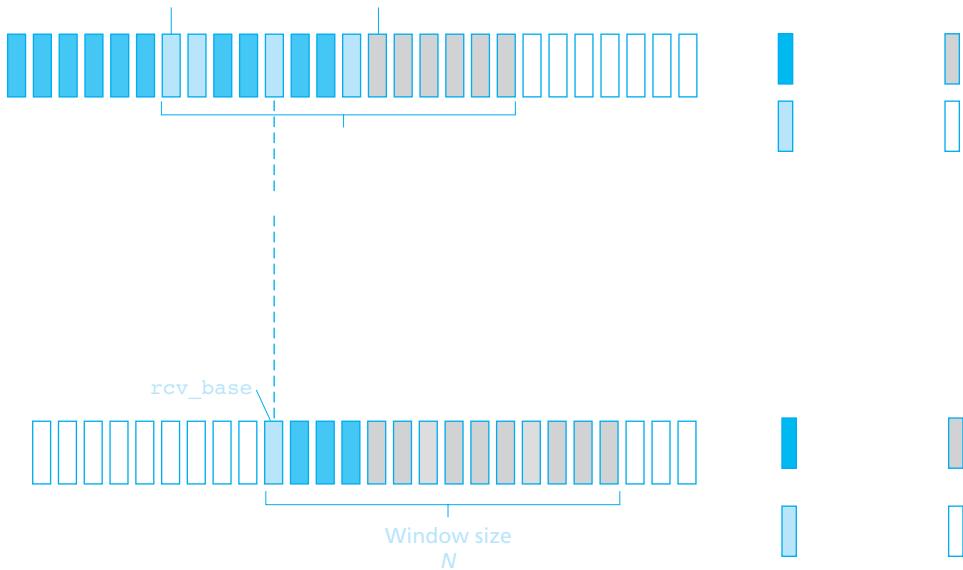


Figure 3.23 ◆ Selective-repeat (SR) sender and receiver views of sequence-number space

of outstanding, unacknowledged packets in the pipeline. However, unlike GBN, the sender will have already received ACKs for some of the packets in the window. Figure 3.23 shows the SR sender’s view of the sequence number space. Figure 3.24 details the various actions taken by the SR sender.

The SR receiver will acknowledge a correctly received packet whether or not it is in order. Out-of-order packets are buffered until any missing packets (that is, packets with lower sequence numbers) are received, at which point a batch of packets can be delivered in order to the upper layer. Figure 3.25 itemizes the various actions taken by the SR receiver. Figure 3.26 shows an example of SR operation in the presence of lost packets. Note that in Figure 3.26, the receiver initially buffers packets 3, 4, and 5, and delivers them together with packet 2 to the upper layer when packet 2 is finally received.

It is important to note that in Step 2 in Figure 3.25, the receiver reacknowledges (rather than ignores) already received packets with certain sequence numbers *below* the current window base. You should convince yourself that this reacknowledgment is indeed needed. Given the sender and receiver sequence number spaces in Figure 3.23, for example, if there is no ACK for packet `send_base` propagating from the receiver to the sender, the sender will eventually retransmit packet `send_base`, even though it is clear (to us, not the sender!) that the receiver has already received

1. *Data received from above.* When data is received from above, the SR sender checks the next available sequence number for the packet. If the sequence number is within the sender's window, the data is packetized and sent; otherwise it is either buffered or returned to the upper layer for later transmission, as in GBN.
2. *Timeout.* Timers are again used to protect against lost packets. However, each packet must now have its own logical timer, since only a single packet will be transmitted on timeout. A single hardware timer can be used to mimic the operation of multiple logical timers [Varghese 1997].
3. *ACK received.* If an ACK is received, the SR sender marks that packet as having been received, provided it is in the window. If the packet's sequence number is equal to `send_base`, the window base is moved forward to the unacknowledged packet with the smallest sequence number. If the window moves and there are untransmitted packets with sequence numbers that now fall within the window, these packets are transmitted.

Figure 3.24 ♦ SR sender events and actions

1. *Packet with sequence number in [`rcv_base`, `rcv_base+N-1`] is correctly received.* In this case, the received packet falls within the receiver's window and a selective ACK packet is returned to the sender. If the packet was not previously received, it is buffered. If this packet has a sequence number equal to the base of the receive window (`rcv_base` in Figure 3.22), then this packet, and any previously buffered and consecutively numbered (beginning with `rcv_base`) packets are delivered to the upper layer. The receive window is then moved forward by the number of packets delivered to the upper layer. As an example, consider Figure 3.26. When a packet with a sequence number of `rcv_base=2` is received, it and packets 3, 4, and 5 can be delivered to the upper layer.
2. *Packet with sequence number in [`rcv_base-N`, `rcv_base-1`] is correctly received.* In this case, an ACK must be generated, even though this is a packet that the receiver has previously acknowledged.
3. *Otherwise.* Ignore the packet.

Figure 3.25 ♦ SR receiver events and actions

that packet. If the receiver were not to acknowledge this packet, the sender's window would never move forward! This example illustrates an important aspect of SR protocols (and many other protocols as well). The sender and receiver will not always have an identical view of what has been received correctly and what has not. For SR protocols, this means that the sender and receiver windows will not always coincide.

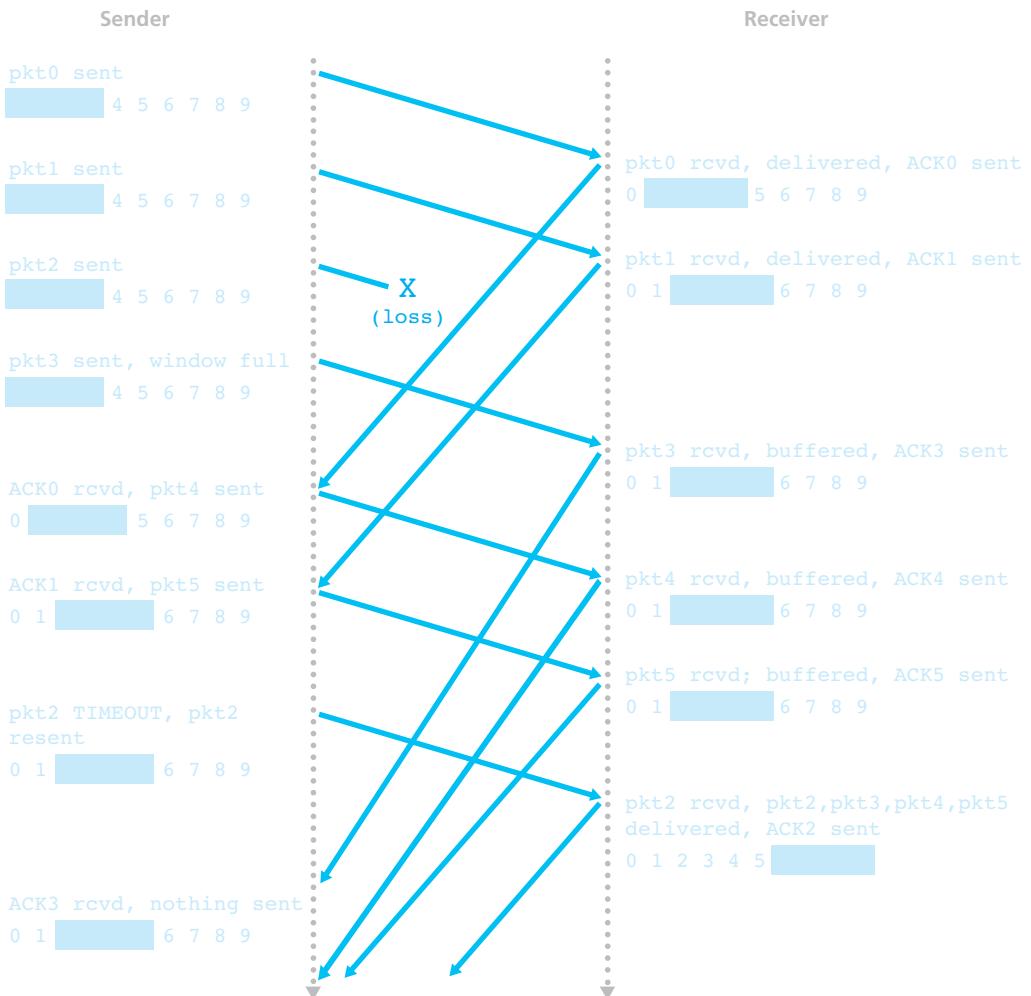


Figure 3.26 ♦ SR operation

The lack of synchronization between sender and receiver windows has important consequences when we are faced with the reality of a finite range of sequence numbers. Consider what could happen, for example, with a finite range of four packet sequence numbers, 0, 1, 2, 3, and a window size of three. Suppose packets 0 through 2 are transmitted and correctly received and acknowledged at the receiver. At this point, the receiver's window is over the fourth, fifth, and sixth packets, which have sequence numbers 3, 0, and 1, respectively. Now consider two scenarios. In the first scenario, shown in Figure 3.27(a), the ACKs for the first three packets are lost and

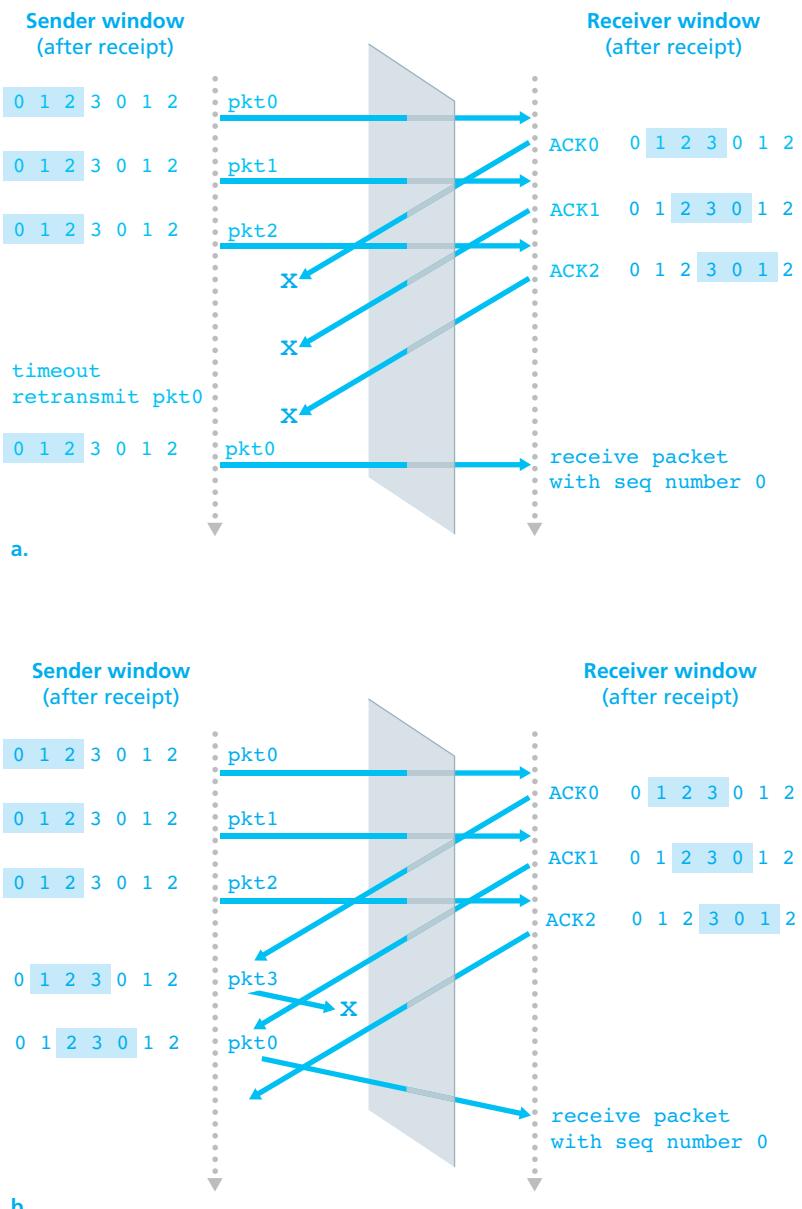


Figure 3.27 ♦ SR receiver dilemma with too-large windows: A new packet or a retransmission?

the sender retransmits these packets. The receiver thus next receives a packet with sequence number 0—a copy of the first packet sent.

In the second scenario, shown in Figure 3.27(b), the ACKs for the first three packets are all delivered correctly. The sender thus moves its window forward and sends the fourth, fifth, and sixth packets, with sequence numbers 3, 0, and 1, respectively. The packet with sequence number 3 is lost, but the packet with sequence number 0 arrives—a packet containing *new* data.

Now consider the receiver's viewpoint in Figure 3.27, which has a figurative curtain between the sender and the receiver, since the receiver cannot "see" the actions taken by the sender. All the receiver observes is the sequence of messages it receives from the channel and sends into the channel. As far as it is concerned, the two scenarios in Figure 3.27 are *identical*. There is no way of distinguishing the retransmission of the first packet from an original transmission of the fifth packet. Clearly, a window size that is 1 less than the size of the sequence number space won't work. But how small must the window size be? A problem at the end of the chapter asks you to show that the window size must be less than or equal to half the size of the sequence number space for SR protocols.

At the companion Web site, you will find an applet that animates the operation of the SR protocol. Try performing the same experiments that you did with the GBN applet. Do the results agree with what you expect?

This completes our discussion of reliable data transfer protocols. We've covered a *lot* of ground and introduced numerous mechanisms that together provide for reliable data transfer. Table 3.1 summarizes these mechanisms. Now that we have seen all of these mechanisms in operation and can see the "big picture," we encourage you to review this section again to see how these mechanisms were incrementally added to cover increasingly complex (and realistic) models of the channel connecting the sender and receiver, or to improve the performance of the protocols.

Let's conclude our discussion of reliable data transfer protocols by considering one remaining assumption in our underlying channel model. Recall that we have assumed that packets cannot be reordered within the channel between the sender and receiver. This is generally a reasonable assumption when the sender and receiver are connected by a single physical wire. However, when the "channel" connecting the two is a network, packet reordering can occur. One manifestation of packet reordering is that old copies of a packet with a sequence or acknowledgment number of x can appear, even though neither the sender's nor the receiver's window contains x . With packet reordering, the channel can be thought of as essentially buffering packets and spontaneously emitting these packets at *any* point in the future. Because sequence numbers may be reused, some care must be taken to guard against such duplicate packets. The approach taken in practice is to ensure that a sequence number is not reused until the sender is "sure" that any previously sent packets with sequence number x are no longer in the network. This is done by assuming that a packet cannot "live" in the network for longer than some fixed maximum amount of time. A maximum packet lifetime of approximately three minutes is assumed in the TCP extensions

Mechanism	Use, Comments
Checksum	Used to detect bit errors in a transmitted packet.
Timer	Used to timeout/retransmit a packet, possibly because the packet (or its ACK) was lost within the channel. Because timeouts can occur when a packet is delayed but not lost (premature timeout), or when a packet has been received by the receiver but the receiver-to-sender ACK has been lost, duplicate copies of a packet may be received by a receiver.
Sequence number	Used for sequential numbering of packets of data flowing from sender to receiver. Gaps in the sequence numbers of received packets allow the receiver to detect a lost packet. Packets with duplicate sequence numbers allow the receiver to detect duplicate copies of a packet.
Acknowledgment	Used by the receiver to tell the sender that a packet or set of packets has been received correctly. Acknowledgments will typically carry the sequence number of the packet or packets being acknowledged. Acknowledgments may be individual or cumulative, depending on the protocol.
Negative acknowledgment	Used by the receiver to tell the sender that a packet has not been received correctly. Negative acknowledgments will typically carry the sequence number of the packet that was not received correctly.
Window, pipelining	The sender may be restricted to sending only packets with sequence numbers that fall within a given range. By allowing multiple packets to be transmitted but not yet acknowledged, sender utilization can be increased over a stop-and-wait mode of operation. We'll see shortly that the window size may be set on the basis of the receiver's ability to receive and buffer messages, or the level of congestion in the network, or both.

Table 3.1 ♦ Summary of reliable data transfer mechanisms and their use

for high-speed networks [RFC 1323]. [Sunshine 1978] describes a method for using sequence numbers such that reordering problems can be completely avoided.

3.5 Connection-Oriented Transport: TCP

Now that we have covered the underlying principles of reliable data transfer, let's turn to TCP—the Internet's transport-layer, connection-oriented, reliable transport protocol. In this section, we'll see that in order to provide reliable data transfer, TCP relies on many of the underlying principles discussed in the previous section, including error detection, retransmissions, cumulative acknowledgments, timers,

and header fields for sequence and acknowledgment numbers. TCP is defined in RFC 793, RFC 1122, RFC 1323, RFC 2018, and RFC 2581.

3.5.1 The TCP Connection

TCP is said to be **connection-oriented** because before one application process can begin to send data to another, the two processes must first “handshake” with each other—that is, they must send some preliminary segments to each other to establish the parameters of the ensuing data transfer. As part of TCP connection establishment, both sides of the connection will initialize many TCP state variables (many of which will be discussed in this section and in Section 3.7) associated with the TCP connection.

The TCP “connection” is not an end-to-end TDM or FDM circuit as in a circuit-switched network. Nor is it a virtual circuit (see Chapter 1), as the connection state resides entirely in the two end systems. Because the TCP protocol runs only in the end systems and not in the intermediate network elements (routers and link-layer switches), the intermediate network elements do not maintain TCP connection state.



CASE HISTORY

VINTON CERF, ROBERT KAHN, AND TCP/IP

In the early 1970s, packet-switched networks began to proliferate, with the ARPAnet—the precursor of the Internet—being just one of many networks. Each of these networks had its own protocol. Two researchers, Vinton Cerf and Robert Kahn, recognized the importance of interconnecting these networks and invented a cross-network protocol called TCP/IP, which stands for Transmission Control Protocol/Internet Protocol. Although Cerf and Kahn began by seeing the protocol as a single entity, it was later split into its two parts, TCP and IP, which operated separately. Cerf and Kahn published a paper on TCP/IP in May 1974 in *IEEE Transactions on Communications Technology* [Cerf 1974].

The TCP/IP protocol, which is the bread and butter of today’s Internet, was devised before PCs, workstations, smartphones, and tablets, before the proliferation of Ethernet, cable, and DSL, WiFi, and other access network technologies, and before the Web, social media, and streaming video. Cerf and Kahn saw the need for a networking protocol that, on the one hand, provides broad support for yet-to-be-defined applications and, on the other hand, allows arbitrary hosts and link-layer protocols to interoperate.

In 2004, Cerf and Kahn received the ACM’s Turing Award, considered the “Nobel Prize of Computing” for “pioneering work on internetworking, including the design and implementation of the Internet’s basic communications protocols, TCP/IP, and for inspired leadership in networking.”

In fact, the intermediate routers are completely oblivious to TCP connections; they see datagrams, not connections.

A TCP connection provides a **full-duplex service**: If there is a TCP connection between Process A on one host and Process B on another host, then application-layer data can flow from Process A to Process B at the same time as application-layer data flows from Process B to Process A. A TCP connection is also always **point-to-point**, that is, between a single sender and a single receiver. So-called “multicasting” (see Section 4.7)—the transfer of data from one sender to many receivers in a single send operation—is not possible with TCP. With TCP, two hosts are company and three are a crowd!

Let’s now take a look at how a TCP connection is established. Suppose a process running in one host wants to initiate a connection with another process in another host. Recall that the process that is initiating the connection is called the *client process*, while the other process is called the *server process*. The client application process first informs the client transport layer that it wants to establish a connection to a process in the server. Recall from Section 2.7.2, a Python client program does this by issuing the command

```
clientSocket.connect((serverName, serverPort))
```

where `serverName` is the name of the server and `serverPort` identifies the process on the server. TCP in the client then proceeds to establish a TCP connection with TCP in the server. At the end of this section we discuss in some detail the connection-establishment procedure. For now it suffices to know that the client first sends a special TCP segment; the server responds with a second special TCP segment; and finally the client responds again with a third special segment. The first two segments carry no payload, that is, no application-layer data; the third of these segments may carry a payload. Because three segments are sent between the two hosts, this connection-establishment procedure is often referred to as a **three-way handshake**.

Once a TCP connection is established, the two application processes can send data to each other. Let’s consider the sending of data from the client process to the server process. The client process passes a stream of data through the socket (the door of the process), as described in Section 2.7. Once the data passes through the door, the data is in the hands of TCP running in the client. As shown in Figure 3.28, TCP directs this data to the connection’s **send buffer**, which is one of the buffers that is set aside during the initial three-way handshake. From time to time, TCP will grab chunks of data from the send buffer and pass the data to the network layer. Interestingly, the TCP specification [RFC 793] is very laid back about specifying when TCP should actually send buffered data, stating that TCP should “send that data in segments at its own convenience.” The maximum amount of data that can be grabbed and placed in a segment is limited by the **maximum segment size (MSS)**. The MSS is typically set by first determining the length of the largest link-layer frame that can be sent by the local sending host (the so-called **maximum**

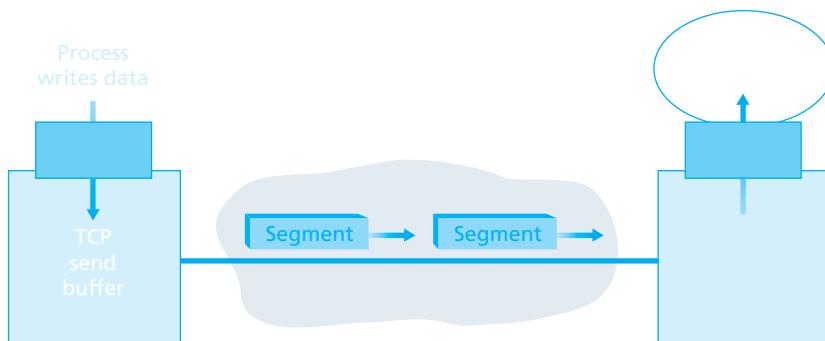


Figure 3.28 ♦ TCP send and receive buffers

transmission unit, MTU), and then setting the MSS to ensure that a TCP segment (when encapsulated in an IP datagram) plus the TCP/IP header length (typically 40 bytes) will fit into a single link-layer frame. Both Ethernet and PPP link-layer protocols have an MSS of 1,500 bytes. Approaches have also been proposed for discovering the path MTU—the largest link-layer frame that can be sent on all links from source to destination [RFC 1191]—and setting the MSS based on the path MTU value. Note that the MSS is the maximum amount of application-layer data in the segment, not the maximum size of the TCP segment including headers. (This terminology is confusing, but we have to live with it, as it is well entrenched.)

TCP pairs each chunk of client data with a TCP header, thereby forming **TCP segments**. The segments are passed down to the network layer, where they are separately encapsulated within network-layer IP datagrams. The IP datagrams are then sent into the network. When TCP receives a segment at the other end, the segment's data is placed in the TCP connection's receive buffer, as shown in Figure 3.28. The application reads the stream of data from this buffer. Each side of the connection has its own send buffer and its own receive buffer. (You can see the online flow-control applet at <http://www.awl.com/kurose-ross>, which provides an animation of the send and receive buffers.)

We see from this discussion that a TCP connection consists of buffers, variables, and a socket connection to a process in one host, and another set of buffers, variables, and a socket connection to a process in another host. As mentioned earlier, no buffers or variables are allocated to the connection in the network elements (routers, switches, and repeaters) between the hosts.

3.5.2 TCP Segment Structure

Having taken a brief look at the TCP connection, let's examine the TCP segment structure. The TCP segment consists of header fields and a data field. The data field contains a chunk of application data. As mentioned above, the MSS limits the

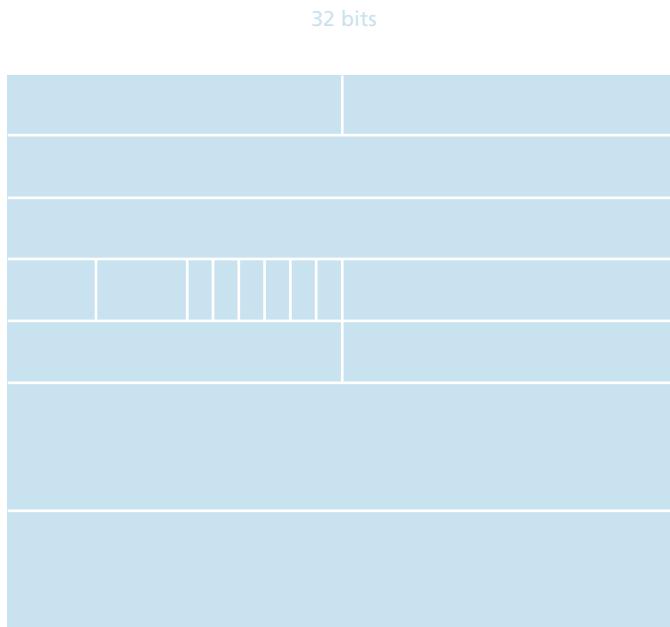


Figure 3.29 ♦ TCP segment structure

maximum size of a segment's data field. When TCP sends a large file, such as an image as part of a Web page, it typically breaks the file into chunks of size MSS (except for the last chunk, which will often be less than the MSS). Interactive applications, however, often transmit data chunks that are smaller than the MSS; for example, with remote login applications like Telnet, the data field in the TCP segment is often only one byte. Because the TCP header is typically 20 bytes (12 bytes more than the UDP header), segments sent by Telnet may be only 21 bytes in length.

Figure 3.29 shows the structure of the TCP segment. As with UDP, the header includes **source and destination port numbers**, which are used for multiplexing/demultiplexing data from/to upper-layer applications. Also, as with UDP, the header includes a **checksum field**. A TCP segment header also contains the following fields:

- The 32-bit **sequence number field** and the 32-bit **acknowledgment number field** are used by the TCP sender and receiver in implementing a reliable data transfer service, as discussed below.
- The 16-bit **receive window field** is used for flow control. We will see shortly that it is used to indicate the number of bytes that a receiver is willing to accept.
- The 4-bit **header length field** specifies the length of the TCP header in 32-bit words. The TCP header can be of variable length due to the TCP options field.

(Typically, the options field is empty, so that the length of the typical TCP header is 20 bytes.)

- The optional and variable-length **options field** is used when a sender and receiver negotiate the maximum segment size (MSS) or as a window scaling factor for use in high-speed networks. A time-stamping option is also defined. See RFC 854 and RFC 1323 for additional details.
- The **flag field** contains 6 bits. The **ACK bit** is used to indicate that the value carried in the acknowledgment field is valid; that is, the segment contains an acknowledgment for a segment that has been successfully received. The **RST**, **SYN**, and **FIN** bits are used for connection setup and teardown, as we will discuss at the end of this section. Setting the **PSH** bit indicates that the receiver should pass the data to the upper layer immediately. Finally, the **URG** bit is used to indicate that there is data in this segment that the sending-side upper-layer entity has marked as “urgent.” The location of the last byte of this urgent data is indicated by the 16-bit **urgent data pointer field**. TCP must inform the receiving-side upper-layer entity when urgent data exists and pass it a pointer to the end of the urgent data. (In practice, the PSH, URG, and the urgent data pointer are not used. However, we mention these fields for completeness.)

Sequence Numbers and Acknowledgment Numbers

Two of the most important fields in the TCP segment header are the sequence number field and the acknowledgment number field. These fields are a critical part of TCP’s reliable data transfer service. But before discussing how these fields are used to provide reliable data transfer, let us first explain what exactly TCP puts in these fields.

TCP views data as an unstructured, but ordered, stream of bytes. TCP’s use of sequence numbers reflects this view in that sequence numbers are over the stream of transmitted bytes and *not* over the series of transmitted segments. The **sequence number for a segment** is therefore the byte-stream number of the first byte in the segment. Let’s look at an example. Suppose that a process in Host A wants to send a stream of data to a process in Host B over a TCP connection. The TCP in Host A will implicitly number each byte in the data stream. Suppose that the data stream consists of a file consisting of 500,000 bytes, that the MSS is 1,000 bytes, and that the first byte of the data stream is numbered 0. As shown in Figure 3.30, TCP constructs 500 segments out of the data stream. The first segment gets assigned sequence number 0, the second segment gets assigned sequence number 1,000, the third segment gets assigned sequence number 2,000, and so on. Each sequence number is inserted in the sequence number field in the header of the appropriate TCP segment.

Now let’s consider acknowledgment numbers. These are a little trickier than sequence numbers. Recall that TCP is full-duplex, so that Host A may be receiving data from Host B while it sends data to Host B (as part of the same TCP connection). Each of the segments that arrive from Host B has a sequence number for the data

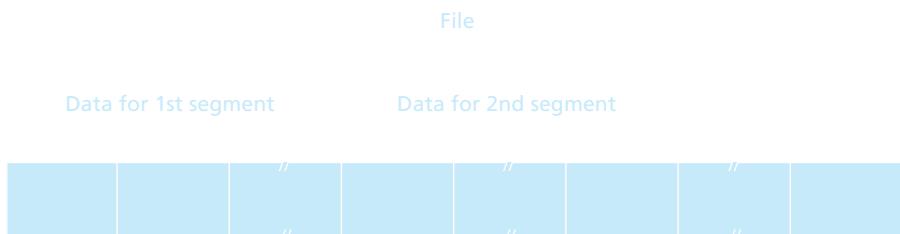


Figure 3.30 ♦ Dividing file data into TCP segments

flowing from B to A. *The acknowledgment number that Host A puts in its segment is the sequence number of the next byte Host A is expecting from Host B.* It is good to look at a few examples to understand what is going on here. Suppose that Host A has received all bytes numbered 0 through 535 from B and suppose that it is about to send a segment to Host B. Host A is waiting for byte 536 and all the subsequent bytes in Host B's data stream. So Host A puts 536 in the acknowledgment number field of the segment it sends to B.

As another example, suppose that Host A has received one segment from Host B containing bytes 0 through 535 and another segment containing bytes 900 through 1,000. For some reason Host A has not yet received bytes 536 through 899. In this example, Host A is still waiting for byte 536 (and beyond) in order to re-create B's data stream. Thus, A's next segment to B will contain 536 in the acknowledgment number field. Because TCP only acknowledges bytes up to the first missing byte in the stream, TCP is said to provide **cumulative acknowledgments**.

This last example also brings up an important but subtle issue. Host A received the third segment (bytes 900 through 1,000) before receiving the second segment (bytes 536 through 899). Thus, the third segment arrived out of order. The subtle issue is: What does a host do when it receives out-of-order segments in a TCP connection? Interestingly, the TCP RFCs do not impose any rules here and leave the decision up to the people programming a TCP implementation. There are basically two choices: either (1) the receiver immediately discards out-of-order segments (which, as we discussed earlier, can simplify receiver design), or (2) the receiver keeps the out-of-order bytes and waits for the missing bytes to fill in the gaps. Clearly, the latter choice is more efficient in terms of network bandwidth, and is the approach taken in practice.

In Figure 3.30, we assumed that the initial sequence number was zero. In truth, both sides of a TCP connection randomly choose an initial sequence number. This is done to minimize the possibility that a segment that is still present in the network from an earlier, already-terminated connection between two hosts is mistaken for a valid segment in a later connection between these same two hosts (which also happen to be using the same port numbers as the old connection) [Sunshine 1978].

Telnet: A Case Study for Sequence and Acknowledgment Numbers

Telnet, defined in RFC 854, is a popular application-layer protocol used for remote login. It runs over TCP and is designed to work between any pair of hosts. Unlike the bulk data transfer applications discussed in Chapter 2, Telnet is an interactive application. We discuss a Telnet example here, as it nicely illustrates TCP sequence and acknowledgment numbers. We note that many users now prefer to use the SSH protocol rather than Telnet, since data sent in a Telnet connection (including passwords!) is not encrypted, making Telnet vulnerable to eavesdropping attacks (as discussed in Section 8.7).

Suppose Host A initiates a Telnet session with Host B. Because Host A initiates the session, it is labeled the client, and Host B is labeled the server. Each character typed by the user (at the client) will be sent to the remote host; the remote host will send back a copy of each character, which will be displayed on the Telnet user's screen. This "echo back" is used to ensure that characters seen by the Telnet user have already been received and processed at the remote site. Each character thus traverses the network twice between the time the user hits the key and the time the character is displayed on the user's monitor.

Now suppose the user types a single letter, 'C,' and then grabs a coffee. Let's examine the TCP segments that are sent between the client and server. As shown in Figure 3.31, we suppose the starting sequence numbers are 42 and 79 for the client and server, respectively. Recall that the sequence number of a segment is the sequence number of the first byte in the data field. Thus, the first segment sent from the client will have sequence number 42; the first segment sent from the server will have sequence number 79. Recall that the acknowledgment number is the sequence number of the next byte of data that the host is waiting for. After the TCP connection is established but before any data is sent, the client is waiting for byte 79 and the server is waiting for byte 42.

As shown in Figure 3.31, three segments are sent. The first segment is sent from the client to the server, containing the 1-byte ASCII representation of the letter 'C' in its data field. This first segment also has 42 in its sequence number field, as we just described. Also, because the client has not yet received any data from the server, this first segment will have 79 in its acknowledgment number field.

The second segment is sent from the server to the client. It serves a dual purpose. First it provides an acknowledgment of the data the server has received. By putting 43 in the acknowledgment field, the server is telling the client that it has successfully received everything up through byte 42 and is now waiting for bytes 43 onward. The second purpose of this segment is to echo back the letter 'C.' Thus, the second segment has the ASCII representation of 'C' in its data field. This second segment has the sequence number 79, the initial sequence number of the server-to-client data flow of this TCP connection, as this is the very first byte of data that the server is sending. Note that the acknowledgment for client-to-server data is carried in a segment carrying server-to-client data; this acknowledgment is said to be **piggybacked** on the server-to-client data segment.

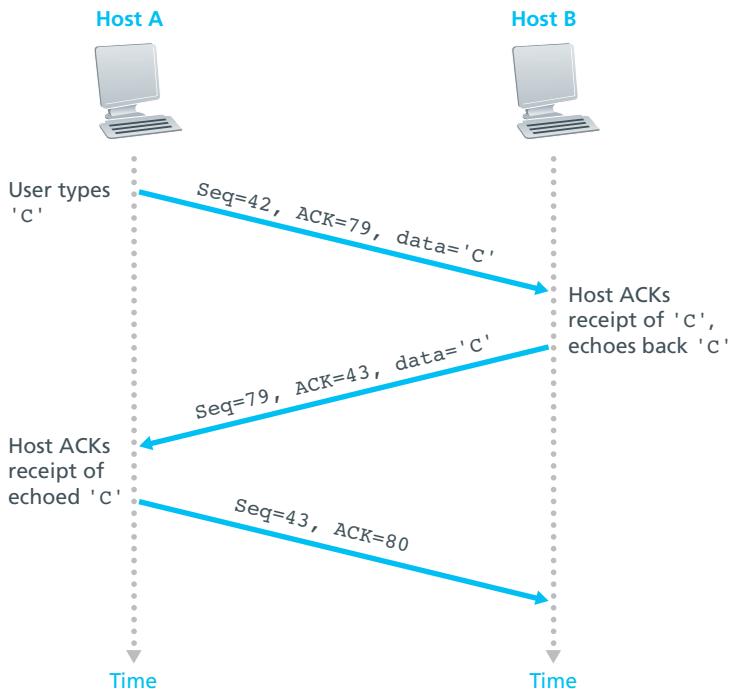


Figure 3.31 ♦ Sequence and acknowledgment numbers for a simple Telnet application over TCP

The third segment is sent from the client to the server. Its sole purpose is to acknowledge the data it has received from the server. (Recall that the second segment contained data—the letter ‘C’—from the server to the client.) This segment has an empty data field (that is, the acknowledgment is not being piggybacked with any client-to-server data). The segment has 80 in the acknowledgment number field because the client has received the stream of bytes up through byte sequence number 79 and it is now waiting for bytes 80 onward. You might think it odd that this segment also has a sequence number since the segment contains no data. But because TCP has a sequence number field, the segment needs to have some sequence number.

3.5.3 Round-Trip Time Estimation and Timeout

TCP, like our `rdt` protocol in Section 3.4, uses a timeout/retransmit mechanism to recover from lost segments. Although this is conceptually simple, many subtle issues arise when we implement a timeout/retransmit mechanism in an actual protocol such as TCP. Perhaps the most obvious question is the length of the timeout

intervals. Clearly, the timeout should be larger than the connection's round-trip time (RTT), that is, the time from when a segment is sent until it is acknowledged. Otherwise, unnecessary retransmissions would be sent. But how much larger? How should the RTT be estimated in the first place? Should a timer be associated with each and every unacknowledged segment? So many questions! Our discussion in this section is based on the TCP work in [Jacobson 1988] and the current IETF recommendations for managing TCP timers [RFC 6298].

Estimating the Round-Trip Time

Let's begin our study of TCP timer management by considering how TCP estimates the round-trip time between sender and receiver. This is accomplished as follows. The sample RTT, denoted `SampleRTT`, for a segment is the amount of time between when the segment is sent (that is, passed to IP) and when an acknowledgement for the segment is received. Instead of measuring a `SampleRTT` for every transmitted segment, most TCP implementations take only one `SampleRTT` measurement at a time. That is, at any point in time, the `SampleRTT` is being estimated for only one of the transmitted but currently unacknowledged segments, leading to a new value of `SampleRTT` approximately once every RTT. Also, TCP never computes a `SampleRTT` for a segment that has been retransmitted; it only measures `SampleRTT` for segments that have been transmitted once [Karn 1987]. (A problem at the end of the chapter asks you to consider why.)

Obviously, the `SampleRTT` values will fluctuate from segment to segment due to congestion in the routers and to the varying load on the end systems. Because of this fluctuation, any given `SampleRTT` value may be atypical. In order to estimate a typical RTT, it is therefore natural to take some sort of average of the `SampleRTT` values. TCP maintains an average, called `EstimatedRTT`, of the `SampleRTT` values. Upon obtaining a new `SampleRTT`, TCP updates `EstimatedRTT` according to the following formula:

$$\text{EstimatedRTT} = (1 - \alpha) \cdot \text{EstimatedRTT} + \alpha \cdot \text{SampleRTT}$$

The formula above is written in the form of a programming-language statement—the new value of `EstimatedRTT` is a weighted combination of the previous value of `EstimatedRTT` and the new value for `SampleRTT`. The recommended value of α is $\alpha = 0.125$ (that is, $1/8$) [RFC 6298], in which case the formula above becomes:

$$\text{EstimatedRTT} = 0.875 \cdot \text{EstimatedRTT} + 0.125 \cdot \text{SampleRTT}$$

Note that `EstimatedRTT` is a weighted average of the `SampleRTT` values. As discussed in a homework problem at the end of this chapter, this weighted average puts more weight on recent samples than on old samples. This is natural, as the



PRINCIPLES IN PRACTICE

TCP provides reliable data transfer by using positive acknowledgments and timers in much the same way that we studied in Section 3.4. TCP acknowledges data that has been received correctly, and it then retransmits segments when segments or their corresponding acknowledgments are thought to be lost or corrupted. Certain versions of TCP also have an implicit NAK mechanism—with TCP’s fast retransmit mechanism, the receipt of three duplicate ACKs for a given segment serves as an implicit NAK for the following segment, triggering retransmission of that segment before timeout. TCP uses sequences of numbers to allow the receiver to identify lost or duplicate segments. Just as in the case of our reliable data transfer protocol, *rdt3.0*, TCP cannot itself tell for certain if a segment, or its ACK, is lost, corrupted, or overly delayed. At the sender, TCP’s response will be the same: retransmit the segment in question.

TCP also uses pipelining, allowing the sender to have multiple transmitted but yet-to-be-acknowledged segments outstanding at any given time. We saw earlier that pipelining can greatly improve a session’s throughput when the ratio of the segment size to round-trip delay is small. The specific number of outstanding, unacknowledged segments that a sender can have is determined by TCP’s flow-control and congestion-control mechanisms. TCP flow control is discussed at the end of this section; TCP congestion control is discussed in Section 3.7. For the time being, we must simply be aware that the TCP sender uses pipelining.

more recent samples better reflect the current congestion in the network. In statistics, such an average is called an **exponential weighted moving average (EWMA)**. The word “exponential” appears in EWMA because the weight of a given `SampleRTT` decays exponentially fast as the updates proceed. In the homework problems you will be asked to derive the exponential term in `EstimatedRTT`.

Figure 3.32 shows the `SampleRTT` values and `EstimatedRTT` for a value of $\alpha = 1/8$ for a TCP connection between `gaia.cs.umass.edu` (in Amherst, Massachusetts) to `fantasia.eurecom.fr` (in the south of France). Clearly, the variations in the `SampleRTT` are smoothed out in the computation of the `EstimatedRTT`.

In addition to having an estimate of the RTT, it is also valuable to have a measure of the variability of the RTT. [RFC 6298] defines the RTT variation, `DevRTT`, as an estimate of how much `SampleRTT` typically deviates from `EstimatedRTT`:

$$\text{DevRTT} = (1 - \beta) \cdot \text{DevRTT} + \beta \cdot |\text{SampleRTT} - \text{EstimatedRTT}|$$

Note that `DevRTT` is an EWMA of the difference between `SampleRTT` and `EstimatedRTT`. If the `SampleRTT` values have little fluctuation, then `DevRTT` will be small; on the other hand, if there is a lot of fluctuation, `DevRTT` will be large. The recommended value of β is 0.25.

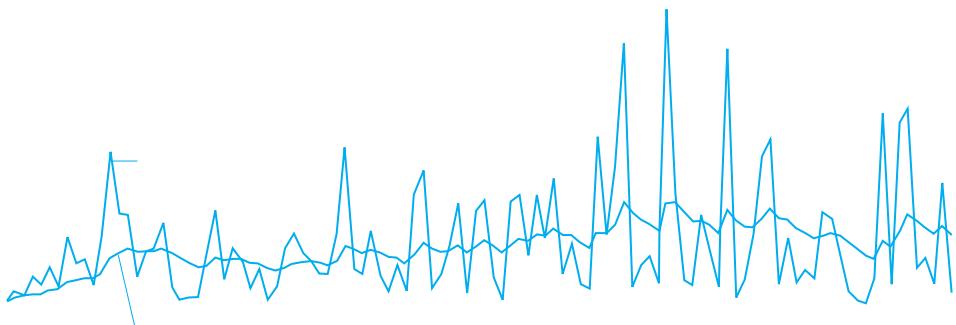


Figure 3.32 ♦ RTT samples and RTT estimates

Setting and Managing the Retransmission Timeout Interval

Given values of `EstimatedRTT` and `DevRTT`, what value should be used for TCP's timeout interval? Clearly, the interval should be greater than or equal to `EstimatedRTT`, or unnecessary retransmissions would be sent. But the timeout interval should not be too much larger than `EstimatedRTT`; otherwise, when a segment is lost, TCP would not quickly retransmit the segment, leading to large data transfer delays. It is therefore desirable to set the timeout equal to the `EstimatedRTT` plus some margin. The margin should be large when there is a lot of fluctuation in the `SampleRTT` values; it should be small when there is little fluctuation. The value of `DevRTT` should thus come into play here. All of these considerations are taken into account in TCP's method for determining the retransmission timeout interval:

```
TimeoutInterval = EstimatedRTT + 4 · DevRTT
```

An initial `TimeoutInterval` value of 1 second is recommended [RFC 6298]. Also, when a timeout occurs, the value of `TimeoutInterval` is doubled to avoid a premature timeout occurring for a subsequent segment that will soon be acknowledged. However, as soon as a segment is received and `EstimatedRTT` is updated, the `TimeoutInterval` is again computed using the formula above.

3.5.4 Reliable Data Transfer

Recall that the Internet's network-layer service (IP service) is unreliable. IP does not guarantee datagram delivery, does not guarantee in-order delivery of datagrams, and does not guarantee the integrity of the data in the datagrams. With IP service, datagrams can overflow router buffers and never reach their destination, datagrams can arrive out of order, and bits in the datagram can get corrupted (flipped from 0 to 1 and vice versa). Because transport-layer segments are carried across the network by IP datagrams, transport-layer segments can suffer from these problems as well.

TCP creates a **reliable data transfer service** on top of IP's unreliable best-effort service. TCP's reliable data transfer service ensures that the data stream that a process reads out of its TCP receive buffer is uncorrupted, without gaps, without duplication, and in sequence; that is, the byte stream is exactly the same byte stream that was sent by the end system on the other side of the connection. How TCP provides a reliable data transfer involves many of the principles that we studied in Section 3.4.

In our earlier development of reliable data transfer techniques, it was conceptually easiest to assume that an individual timer is associated with each transmitted but not yet acknowledged segment. While this is great in theory, timer management can require considerable overhead. Thus, the recommended TCP timer management procedures [RFC 6298] use only a *single* retransmission timer, even if there are multiple transmitted but not yet acknowledged segments. The TCP protocol described in this section follows this single-timer recommendation.

We will discuss how TCP provides reliable data transfer in two incremental steps. We first present a highly simplified description of a TCP sender that uses only timeouts to recover from lost segments; we then present a more complete description that uses duplicate acknowledgments in addition to timeouts. In the ensuing discussion, we suppose that data is being sent in only one direction, from Host A to Host B, and that Host A is sending a large file.

Figure 3.33 presents a highly simplified description of a TCP sender. We see that there are three major events related to data transmission and retransmission in the TCP sender: data received from application above; timer timeout; and ACK receipt. Upon the occurrence of the first major event, TCP receives data from the application, encapsulates the data in a segment, and passes the segment to IP. Note that each segment includes a sequence number that is the byte-stream number of the first data byte in the segment, as described in Section 3.5.2. Also note that if the timer is already not running for some other segment, TCP starts the timer when the segment is passed to IP. (It is helpful to think of the timer as being associated with the oldest unacknowledged segment.) The expiration interval for this timer is the `TimeoutInterval`, which is calculated from `EstimatedRTT` and `DevRTT`, as described in Section 3.5.3.

```
/* Assume sender is not constrained by TCP flow or congestion control, that data from above is less
than MSS in size, and that data transfer is in one direction only. */

NextSeqNum=InitialSeqNumber
SendBase=InitialSeqNumber

loop (forever) {
    switch(event)

        event: data received from application above
            create TCP segment with sequence number NextSeqNum
            if (timer currently not running)
                start timer
            pass segment to IP
            NextSeqNum=NextSeqNum+length(data)
            break;

        event: timer timeout
            retransmit not-yet-acknowledged segment with
                smallest sequence number
            start timer
            break;

        event: ACK received, with ACK field value of y
            if (y > SendBase) {
                SendBase=y
                if (there are currently any not-yet-acknowledged segments)
                    start timer
            }
            break;

    } /* end of loop forever */
}
```

Figure 3.33 ♦ Simplified TCP sender

The second major event is the timeout. TCP responds to the timeout event by retransmitting the segment that caused the timeout. TCP then restarts the timer.

The third major event that must be handled by the TCP sender is the arrival of an acknowledgment segment (ACK) from the receiver (more specifically, a segment containing a valid ACK field value). On the occurrence of this event, TCP compares the ACK value y with its variable `SendBase`. The TCP state variable `SendBase` is the sequence number of the oldest unacknowledged byte. (Thus $SendBase-1$ is the sequence number of the last byte that is known to have been received correctly and in order at the receiver.) As indicated earlier, TCP uses cumulative acknowledgments, so that y acknowledges the receipt of all bytes before byte number y . If $y > SendBase$,

then the ACK is acknowledging one or more previously unacknowledged segments. Thus the sender updates its `SendBase` variable; it also restarts the timer if there currently are any not-yet-acknowledged segments.

A Few Interesting Scenarios

We have just described a highly simplified version of how TCP provides reliable data transfer. But even this highly simplified version has many subtleties. To get a good feeling for how this protocol works, let's now walk through a few simple scenarios. Figure 3.34 depicts the first scenario, in which Host A sends one segment to Host B. Suppose that this segment has sequence number 92 and contains 8 bytes of data. After sending this segment, Host A waits for a segment from B with acknowledgment number 100. Although the segment from A is received at B, the acknowledgment from B to A gets lost. In this case, the timeout event occurs, and Host A retransmits the same segment. Of course, when Host B receives the retransmission, it observes from the sequence number that the segment contains data that has already been received. Thus, TCP in Host B will discard the bytes in the retransmitted segment.

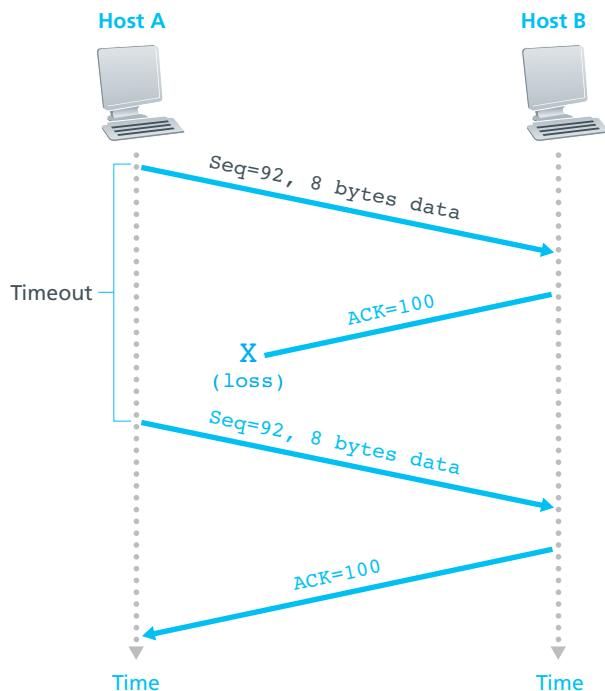


Figure 3.34 ♦ Retransmission due to a lost acknowledgement

In a second scenario, shown in Figure 3.35, Host A sends two segments back to back. The first segment has sequence number 92 and 8 bytes of data, and the second segment has sequence number 100 and 20 bytes of data. Suppose that both segments arrive intact at B, and B sends two separate acknowledgments for each of these segments. The first of these acknowledgments has acknowledgment number 100; the second has acknowledgment number 120. Suppose now that neither of the acknowledgments arrives at Host A before the timeout. When the timeout event occurs, Host A resends the first segment with sequence number 92 and restarts the timer. As long as the ACK for the second segment arrives before the new timeout, the second segment will not be retransmitted.

In a third and final scenario, suppose Host A sends the two segments, exactly as in the second example. The acknowledgment of the first segment is lost in the network, but just before the timeout event, Host A receives an acknowledgment with acknowledgment number 120. Host A therefore knows that Host B has received *everything* up through byte 119; so Host A does not resend either of the two segments. This scenario is illustrated in Figure 3.36.

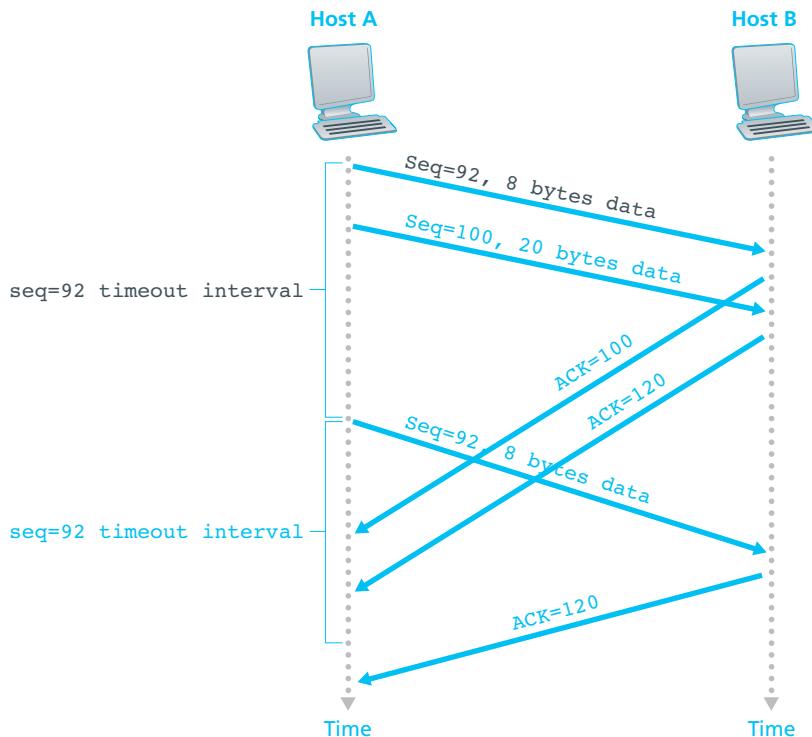


Figure 3.35 ♦ Segment 100 not retransmitted

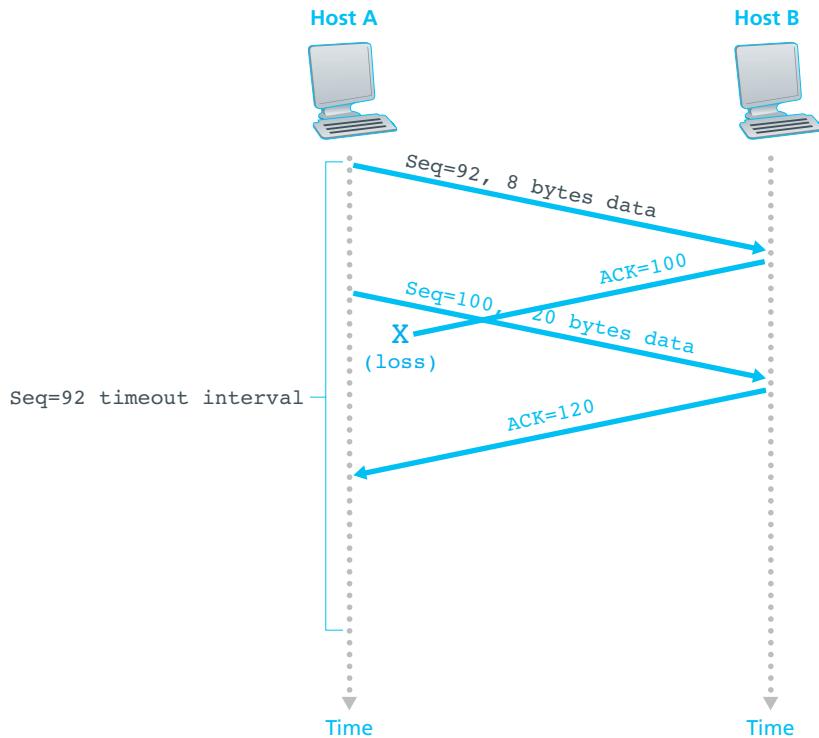


Figure 3.36 ♦ A cumulative acknowledgment avoids retransmission of the first segment

Doubling the Timeout Interval

We now discuss a few modifications that most TCP implementations employ. The first concerns the length of the timeout interval after a timer expiration. In this modification, whenever the timeout event occurs, TCP retransmits the not-yet-acknowledged segment with the smallest sequence number, as described above. But each time TCP retransmits, it sets the next timeout interval to twice the previous value, rather than deriving it from the last `EstimatedRTT` and `DevRTT` (as described in Section 3.5.3). For example, suppose `TimeoutInterval` associated with the oldest not yet acknowledged segment is .75 sec when the timer first expires. TCP will then retransmit this segment and set the new expiration time to 1.5 sec. If the timer expires again 1.5 sec later, TCP will again retransmit this segment, now setting the expiration time to 3.0 sec. Thus the intervals grow exponentially after each retransmission. However, whenever the timer is started after either of the two other events (that is, data received from application above, and ACK received), the

`TimeoutInterval` is derived from the most recent values of `EstimatedRTT` and `DevRTT`.

This modification provides a limited form of congestion control. (More comprehensive forms of TCP congestion control will be studied in Section 3.7.) The timer expiration is most likely caused by congestion in the network, that is, too many packets arriving at one (or more) router queues in the path between the source and destination, causing packets to be dropped and/or long queuing delays. In times of congestion, if the sources continue to retransmit packets persistently, the congestion may get worse. Instead, TCP acts more politely, with each sender retransmitting after longer and longer intervals. We will see that a similar idea is used by Ethernet when we study CSMA/CD in Chapter 5.

Fast Retransmit

One of the problems with timeout-triggered retransmissions is that the timeout period can be relatively long. When a segment is lost, this long timeout period forces the sender to delay resending the lost packet, thereby increasing the end-to-end delay. Fortunately, the sender can often detect packet loss well before the timeout event occurs by noting so-called duplicate ACKs. A **duplicate ACK** is an ACK that reacknowledges a segment for which the sender has already received an earlier acknowledgment. To understand the sender's response to a duplicate ACK, we must look at why the receiver sends a duplicate ACK in the first place. Table 3.2 summarizes the TCP receiver's ACK generation policy [RFC 5681]. When a TCP receiver receives a segment with a sequence number that is larger than the next, expected, in-order sequence number, it detects a gap in the data stream—that is, a missing segment. This gap could be the result of lost or reordered segments within the network.

Event	TCP Receiver Action
Arrival of in-order segment with expected sequence number. All data up to expected sequence number already acknowledged.	Delayed ACK. Wait up to 500 msec for arrival of another in-order segment. If next in-order segment does not arrive in this interval, send an ACK.
Arrival of in-order segment with expected sequence number. One other in-order segment waiting for ACK transmission.	Immediately send single cumulative ACK, ACKing both in-order segments.
Arrival of out-of-order segment with higher-than-expected sequence number. Gap detected.	Immediately send duplicate ACK, indicating sequence number of next expected byte (which is the lower end of the gap).
Arrival of segment that partially or completely fills in gap in received data.	Immediately send ACK, provided that segment starts at the lower end of gap.

Table 3.2 ♦ TCP ACK Generation Recommendation [RFC 5681]

Since TCP does not use negative acknowledgments, the receiver cannot send an explicit negative acknowledgment back to the sender. Instead, it simply reacknowledges (that is, generates a duplicate ACK for) the last in-order byte of data it has received. (Note that Table 3.2 allows for the case that the receiver does not discard out-of-order segments.)

Because a sender often sends a large number of segments back to back, if one segment is lost, there will likely be many back-to-back duplicate ACKs. If the TCP sender receives three duplicate ACKs for the same data, it takes this as an indication that the segment following the segment that has been ACKed three times has been lost. (In the homework problems, we consider the question of why the sender waits for three duplicate ACKs, rather than just a single duplicate ACK.) In the case that three duplicate ACKs are received, the TCP sender performs a **fast retransmit** [RFC 5681], retransmitting the missing segment *before* that segment's timer expires. This is shown in Figure 3.37, where the second segment is lost, then retransmitted before its timer expires. For TCP with fast retransmit, the following code snippet replaces the ACK received event in Figure 3.33:

```
event: ACK received, with ACK field value of y
    if (y > SendBase) {
        SendBase=y
        if (there are currently any not yet
            acknowledged segments)
            start timer
    }
    else { /* a duplicate ACK for already ACKed
        segment */
        increment number of duplicate ACKs
        received for y
        if (number of duplicate ACKS received
            for y==3)
            /* TCP fast retransmit */
            resend segment with sequence number y
    }
break;
```

We noted earlier that many subtle issues arise when a timeout/retransmit mechanism is implemented in an actual protocol such as TCP. The procedures above, which have evolved as a result of more than 20 years of experience with TCP timers, should convince you that this is indeed the case!

Go-Back-N or Selective Repeat?

Let us close our study of TCP's error-recovery mechanism by considering the following question: Is TCP a GBN or an SR protocol? Recall that TCP acknowledgments are cumulative and correctly received but out-of-order segments are not individually

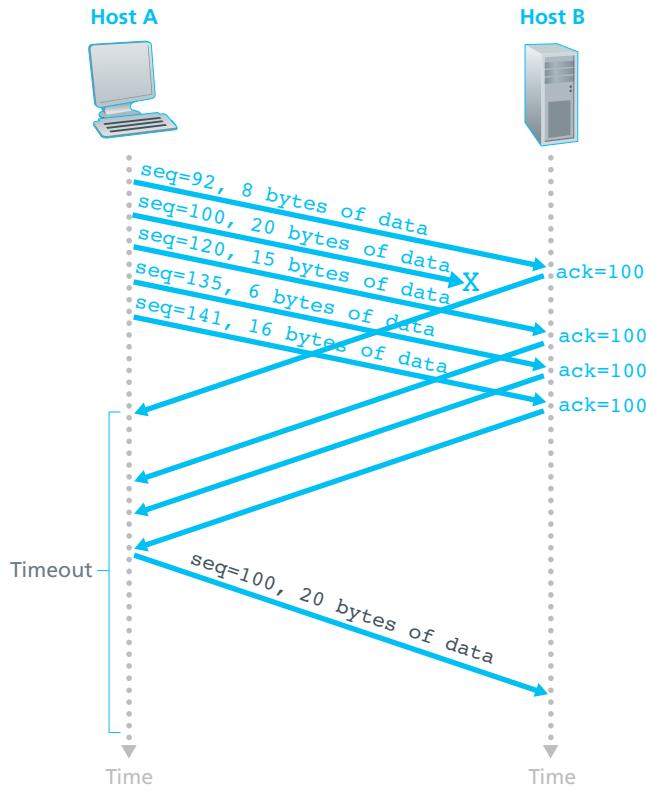


Figure 3.37 ♦ Fast retransmit: retransmitting the missing segment before the segment's timer expires

ACKed by the receiver. Consequently, as shown in Figure 3.33 (see also Figure 3.19), the TCP sender need only maintain the smallest sequence number of a transmitted but unacknowledged byte (`SendBase`) and the sequence number of the next byte to be sent (`NextSeqNum`). In this sense, TCP looks a lot like a GBN-style protocol. But there are some striking differences between TCP and Go-Back-N. Many TCP implementations will buffer correctly received but out-of-order segments [Stevens 1994]. Consider also what happens when the sender sends a sequence of segments $1, 2, \dots, N$, and all of the segments arrive in order without error at the receiver. Further suppose that the acknowledgment for packet $n < N$ gets lost, but the remaining $N - 1$ acknowledgments arrive at the sender before their respective timeouts. In this example, GBN would retransmit not only packet n , but also all of the subsequent packets $n + 1, n + 2, \dots, N$. TCP, on the other hand, would retransmit at most one segment, namely, segment n . Moreover, TCP would not even retransmit segment n if the acknowledgment for segment $n + 1$ arrived before the timeout for segment n .

A proposed modification to TCP, the so-called **selective acknowledgment** [RFC 2018], allows a TCP receiver to acknowledge out-of-order segments selectively rather than just cumulatively acknowledging the last correctly received, in-order segment. When combined with selective retransmission—skipping the retransmission of segments that have already been selectively acknowledged by the receiver—TCP looks a lot like our generic SR protocol. Thus, TCP’s error-recovery mechanism is probably best categorized as a hybrid of GBN and SR protocols.

3.5.5 Flow Control

Recall that the hosts on each side of a TCP connection set aside a receive buffer for the connection. When the TCP connection receives bytes that are correct and in sequence, it places the data in the receive buffer. The associated application process will read data from this buffer, but not necessarily at the instant the data arrives. Indeed, the receiving application may be busy with some other task and may not even attempt to read the data until long after it has arrived. If the application is relatively slow at reading the data, the sender can very easily overflow the connection’s receive buffer by sending too much data too quickly.

TCP provides a **flow-control service** to its applications to eliminate the possibility of the sender overflowing the receiver’s buffer. Flow control is thus a speed-matching service—matching the rate at which the sender is sending against the rate at which the receiving application is reading. As noted earlier, a TCP sender can also be throttled due to congestion within the IP network; this form of sender control is referred to as **congestion control**, a topic we will explore in detail in Sections 3.6 and 3.7. Even though the actions taken by flow and congestion control are similar (the throttling of the sender), they are obviously taken for very different reasons. Unfortunately, many authors use the terms interchangeably, and the savvy reader would be wise to distinguish between them. Let’s now discuss how TCP provides its flow-control service. In order to see the forest for the trees, we suppose throughout this section that the TCP implementation is such that the TCP receiver discards out-of-order segments.

TCP provides flow control by having the *sender* maintain a variable called the **receive window**. Informally, the receive window is used to give the sender an idea of how much free buffer space is available at the receiver. Because TCP is full-duplex, the sender at each side of the connection maintains a distinct receive window. Let’s investigate the receive window in the context of a file transfer. Suppose that Host A is sending a large file to Host B over a TCP connection. Host B allocates a receive buffer to this connection; denote its size by $RcvBuffer$. From time to time, the application process in Host B reads from the buffer. Define the following variables:

- **LastByteRead**: the number of the last byte in the data stream read from the buffer by the application process in B
- **LastByteRcvd**: the number of the last byte in the data stream that has arrived from the network and has been placed in the receive buffer at B

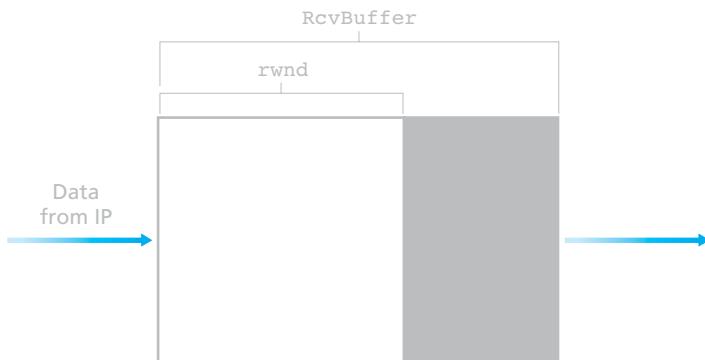


Figure 3.38 ♦ The receive window (`rwnd`) and the receive buffer (`RcvBuffer`)

Because TCP is not permitted to overflow the allocated buffer, we must have

$$\text{LastByteRcvd} - \text{LastByteRead} \leq \text{RcvBuffer}$$

The receive window, denoted `rwnd` is set to the amount of spare room in the buffer:

$$\text{rwnd} = \text{RcvBuffer} - [\text{LastByteRcvd} - \text{LastByteRead}]$$

Because the spare room changes with time, `rwnd` is dynamic. The variable `rwnd` is illustrated in Figure 3.38.

How does the connection use the variable `rwnd` to provide the flow-control service? Host B tells Host A how much spare room it has in the connection buffer by placing its current value of `rwnd` in the receive window field of every segment it sends to A. Initially, Host B sets `rwnd = RcvBuffer`. Note that to pull this off, Host B must keep track of several connection-specific variables.

Host A in turn keeps track of two variables, `LastByteSent` and `LastByteAcked`, which have obvious meanings. Note that the difference between these two variables, `LastByteSent - LastByteAcked`, is the amount of unacknowledged data that A has sent into the connection. By keeping the amount of unacknowledged data less than the value of `rwnd`, Host A is assured that it is not overflowing the receive buffer at Host B. Thus, Host A makes sure throughout the connection's life that

$$\text{LastByteSent} - \text{LastByteAcked} \leq \text{rwnd}$$

There is one minor technical problem with this scheme. To see this, suppose Host B's receive buffer becomes full so that $rwnd = 0$. After advertising $rwnd = 0$ to Host A, also suppose that B has *nothing* to send to A. Now consider what happens. As the application process at B empties the buffer, TCP does not send new segments with new $rwnd$ values to Host A; indeed, TCP sends a segment to Host A only if it has data to send or if it has an acknowledgment to send. Therefore, Host A is never informed that some space has opened up in Host B's receive buffer—Host A is blocked and can transmit no more data! To solve this problem, the TCP specification requires Host A to continue to send segments with one data byte when B's receive window is zero. These segments will be acknowledged by the receiver. Eventually the buffer will begin to empty and the acknowledgments will contain a nonzero $rwnd$ value.

The online site at <http://www.awl.com/kurose-ross> for this book provides an interactive Java applet that illustrates the operation of the TCP receive window.

Having described TCP's flow-control service, we briefly mention here that UDP does not provide flow control. To understand the issue, consider sending a series of UDP segments from a process on Host A to a process on Host B. For a typical UDP implementation, UDP will append the segments in a finite-sized buffer that "precedes" the corresponding socket (that is, the door to the process). The process reads one entire segment at a time from the buffer. If the process does not read the segments fast enough from the buffer, the buffer will overflow and segments will get dropped.

3.5.6 TCP Connection Management

In this subsection we take a closer look at how a TCP connection is established and torn down. Although this topic may not seem particularly thrilling, it is important because TCP connection establishment can significantly add to perceived delays (for example, when surfing the Web). Furthermore, many of the most common network attacks—including the incredibly popular SYN flood attack—exploit vulnerabilities in TCP connection management. Let's first take a look at how a TCP connection is established. Suppose a process running in one host (client) wants to initiate a connection with another process in another host (server). The client application process first informs the client TCP that it wants to establish a connection to a process in the server. The TCP in the client then proceeds to establish a TCP connection with the TCP in the server in the following manner:

- *Step 1.* The client-side TCP first sends a special TCP segment to the server-side TCP. This special segment contains no application-layer data. But one of the flag bits in the segment's header (see Figure 3.29), the SYN bit, is set to 1. For this reason, this special segment is referred to as a SYN segment. In addition, the client randomly chooses an initial sequence number (`client_isn`) and puts this number in the sequence number field of the initial TCP SYN segment. This segment is encapsulated within an IP datagram and sent to the server. There has

been considerable interest in properly randomizing the choice of the `client_isn` in order to avoid certain security attacks [CERT 2001–09].

- *Step 2.* Once the IP datagram containing the TCP SYN segment arrives at the server host (assuming it does arrive!), the server extracts the TCP SYN segment from the datagram, allocates the TCP buffers and variables to the connection, and sends a connection-granted segment to the client TCP. (We'll see in Chapter 8 that the allocation of these buffers and variables before completing the third step of the three-way handshake makes TCP vulnerable to a denial-of-service attack known as SYN flooding.) This connection-granted segment also contains no application-layer data. However, it does contain three important pieces of information in the segment header. First, the SYN bit is set to 1. Second, the acknowledgment field of the TCP segment header is set to `client_isn+1`. Finally, the server chooses its own initial sequence number (`server_isn`) and puts this value in the sequence number field of the TCP segment header. This connection-granted segment is saying, in effect, "I received your SYN packet to start a connection with your initial sequence number, `client_isn`. I agree to establish this connection. My own initial sequence number is `server_isn`." The connection-granted segment is referred to as a **SYNACK segment**.
- *Step 3.* Upon receiving the SYNACK segment, the client also allocates buffers and variables to the connection. The client host then sends the server yet another segment; this last segment acknowledges the server's connection-granted segment (the client does so by putting the value `server_isn+1` in the acknowledgment field of the TCP segment header). The SYN bit is set to zero, since the connection is established. This third stage of the three-way handshake may carry client-to-server data in the segment payload.

Once these three steps have been completed, the client and server hosts can send segments containing data to each other. In each of these future segments, the SYN bit will be set to zero. Note that in order to establish the connection, three packets are sent between the two hosts, as illustrated in Figure 3.39. For this reason, this connection-establishment procedure is often referred to as a **three-way handshake**. Several aspects of the TCP three-way handshake are explored in the homework problems (Why are initial sequence numbers needed? Why is a three-way handshake, as opposed to a two-way handshake, needed?). It's interesting to note that a rock climber and a belayer (who is stationed below the rock climber and whose job it is to handle the climber's safety rope) use a three-way-handshake communication protocol that is identical to TCP's to ensure that both sides are ready before the climber begins ascent.

All good things must come to an end, and the same is true with a TCP connection. Either of the two processes participating in a TCP connection can end the connection. When a connection ends, the "resources" (that is, the buffers and variables) in the hosts are deallocated. As an example, suppose the client decides to close the connection, as shown in Figure 3.40. The client application process issues a close

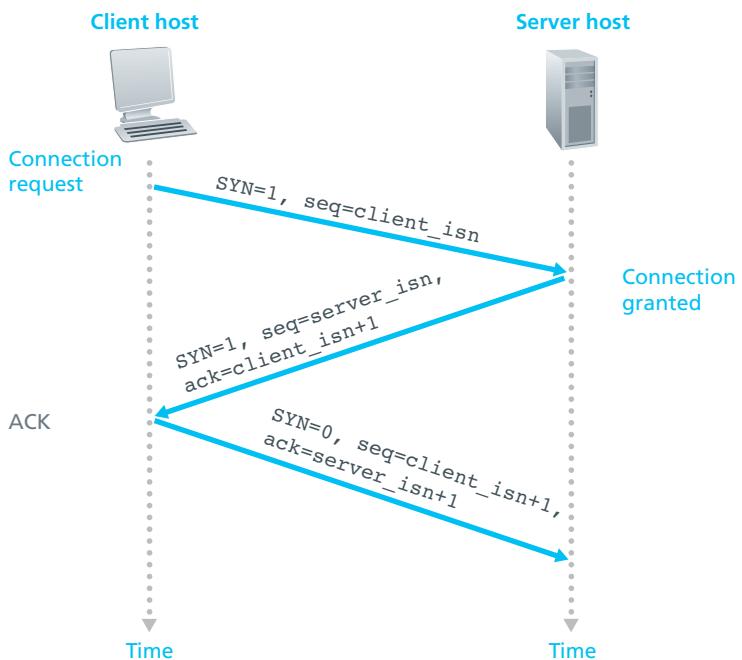


Figure 3.39 ♦ TCP three-way handshake: segment exchange

command. This causes the client TCP to send a special TCP segment to the server process. This special segment has a flag bit in the segment's header, the FIN bit (see Figure 3.29), set to 1. When the server receives this segment, it sends the client an acknowledgment segment in return. The server then sends its own shutdown segment, which has the FIN bit set to 1. Finally, the client acknowledges the server's shutdown segment. At this point, all the resources in the two hosts are now deallocated.

During the life of a TCP connection, the TCP protocol running in each host makes transitions through various **TCP states**. Figure 3.41 illustrates a typical sequence of TCP states that are visited by the *client* TCP. The client TCP begins in the CLOSED state. The application on the client side initiates a new TCP connection (by creating a Socket object in our Java examples as in the Python examples from Chapter 2). This causes TCP in the client to send a SYN segment to TCP in the server. After having sent the SYN segment, the client TCP enters the SYN_SENT state. While in the SYN_SENT state, the client TCP waits for a segment from the server TCP that includes an acknowledgment for the client's previous segment and has the SYN bit set to 1. Having received such a segment, the client TCP enters the ESTABLISHED state. While in the ESTABLISHED state, the TCP client can send and receive TCP segments containing payload (that is, application-generated) data.

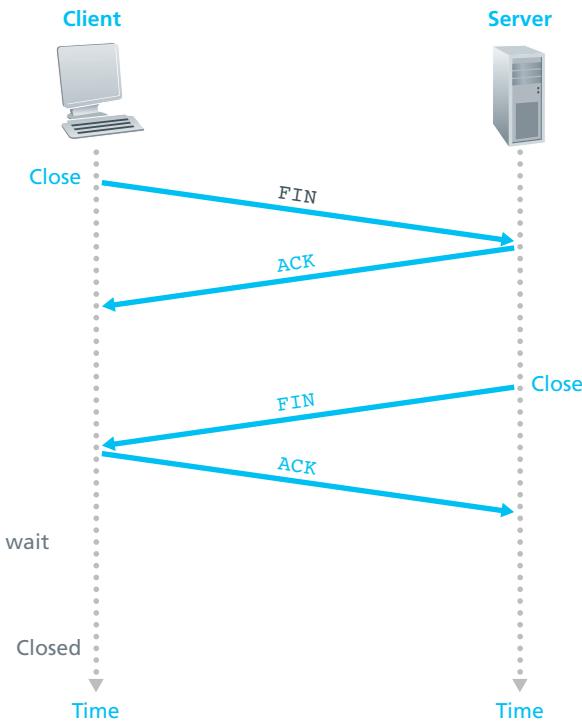


Figure 3.40 ♦ Closing a TCP connection

Suppose that the client application decides it wants to close the connection. (Note that the server could also choose to close the connection.) This causes the client TCP to send a TCP segment with the FIN bit set to 1 and to enter the FIN_WAIT_1 state. While in the FIN_WAIT_1 state, the client TCP waits for a TCP segment from the server with an acknowledgment. When it receives this segment, the client TCP enters the FIN_WAIT_2 state. While in the FIN_WAIT_2 state, the client waits for another segment from the server with the FIN bit set to 1; after receiving this segment, the client TCP acknowledges the server's segment and enters the TIME_WAIT state. The TIME_WAIT state lets the TCP client resend the final acknowledgment in case the ACK is lost. The time spent in the TIME_WAIT state is implementation-dependent, but typical values are 30 seconds, 1 minute, and 2 minutes. After the wait, the connection formally closes and all resources on the client side (including port numbers) are released.

Figure 3.42 illustrates the series of states typically visited by the server-side TCP, assuming the client begins connection teardown. The transitions are self-explanatory. In these two state-transition diagrams, we have only shown how a TCP connection is normally established and shut down. We have not described what

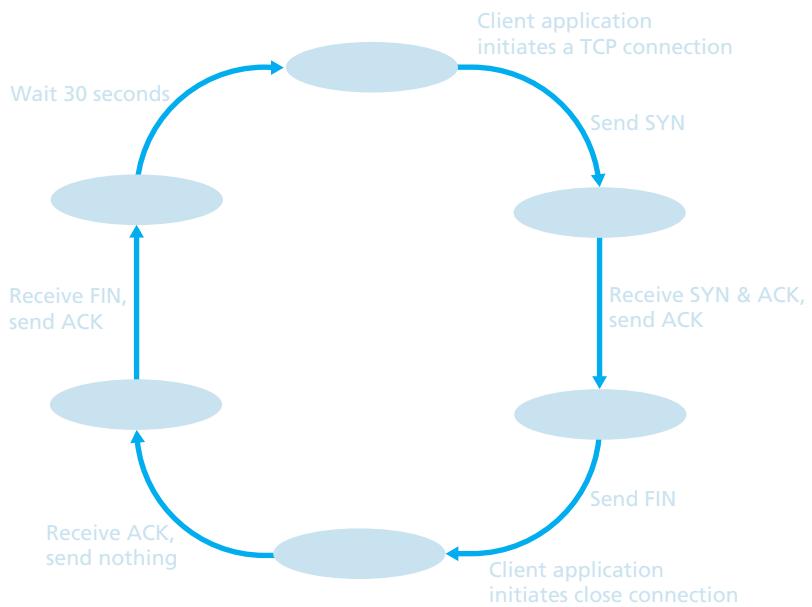


Figure 3.41 ♦ A typical sequence of TCP states visited by a client TCP

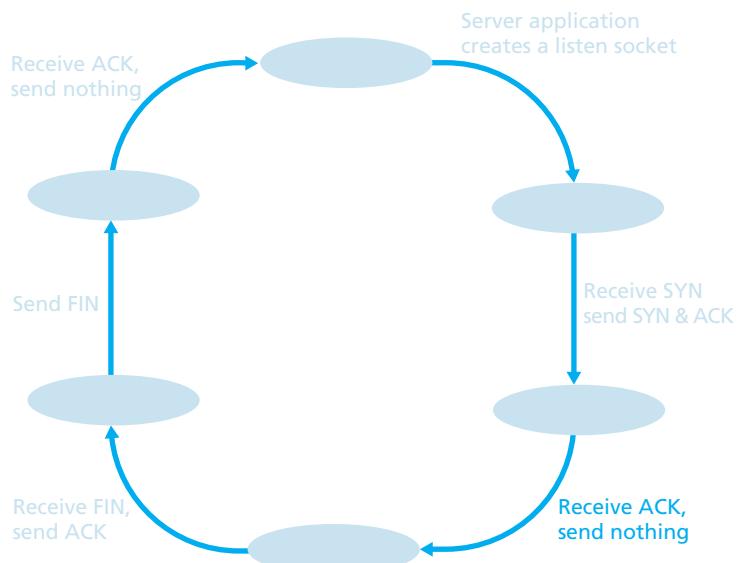


Figure 3.42 ♦ A typical sequence of TCP states visited by a server-side TCP



FOCUS ON SECURITY

THE SYN FLOOD ATTACK

We've seen in our discussion of TCP's three-way handshake that a server allocates and initializes connection variables and buffers in response to a received SYN. The server then sends a SYNACK in response, and awaits an ACK segment from the client. If the client does not send an ACK to complete the third step of this 3-way handshake, eventually (often after a minute or more) the server will terminate the half-open connection and reclaim the allocated resources.

This TCP connection management protocol sets the stage for a classic Denial of Service (DoS) attack known as the **SYN flood attack**. In this attack, the attacker(s) send a large number of TCP SYN segments, without completing the third handshake step. With this deluge of SYN segments, the server's connection resources become exhausted as they are allocated (but never used!) for half-open connections; legitimate clients are then denied service. Such SYN flooding attacks were among the first documented DoS attacks [CERT SYN 1996]. Fortunately, an effective defense known as **SYN cookies** [RFC 4987] are now deployed in most major operating systems. SYN cookies work as follows:

- o When the server receives a SYN segment, it does not know if the segment is coming from a legitimate user or is part of a SYN flood attack. So, instead of creating a half-open TCP connection for this SYN, the server creates an initial TCP sequence number that is a complicated function (hash function) of source and destination IP addresses and port numbers of the SYN segment, as well as a secret number only known to the server. This carefully crafted initial sequence number is the so-called "cookie." The server then sends the client a SYNACK packet with this special initial sequence number. *Importantly, the server does not remember the cookie or any other state information corresponding to the SYN.*
- o A legitimate client will return an ACK segment. When the server receives this ACK, it must verify that the ACK corresponds to some SYN sent earlier. But how is this done if the server maintains no memory about SYN segments? As you may have guessed, it is done with the cookie. Recall that for a legitimate ACK, the value in the acknowledgment field is equal to the initial sequence number in the SYNACK (the cookie value in this case) plus one (see Figure 3.39). The server can then run the same hash function using the source and destination IP address and port numbers in the SYNACK (which are the same as in the original SYN) and the secret number. If the result of the function plus one is the same as the acknowledgment (cookie) value in the client's SYNACK, the server concludes that the ACK corresponds to an earlier SYN segment and is hence valid. The server then creates a fully open connection along with a socket.
- o On the other hand, if the client does not return an ACK segment, then the original SYN has done no harm at the server, since the server hasn't yet allocated any resources in response to the original bogus SYN.

happens in certain pathological scenarios, for example, when both sides of a connection want to initiate or shut down at the same time. If you are interested in learning about this and other advanced issues concerning TCP, you are encouraged to see Stevens' comprehensive book [Stevens 1994].

Our discussion above has assumed that both the client and server are prepared to communicate, i.e., that the server is listening on the port to which the client sends its SYN segment. Let's consider what happens when a host receives a TCP segment whose port numbers or source IP address do not match with any of the ongoing sockets in the host. For example, suppose a host receives a TCP SYN packet with destination port 80, but the host is not accepting connections on port 80 (that is, it is not running a Web server on port 80). Then the host will send a special reset segment to the source. This TCP segment has the RST flag bit (see Section 3.5.2) set to 1. Thus, when a host sends a reset segment, it is telling the source "I don't have a socket for that segment. Please do not resend the segment." When a host receives a UDP packet whose destination port number doesn't match with an ongoing UDP socket, the host sends a special ICMP datagram, as discussed in Chapter 4.

Now that we have a good understanding of TCP connection management, let's revisit the nmap port-scanning tool and examine more closely how it works. To explore a specific TCP port, say port 6789, on a target host, nmap will send a TCP SYN segment with destination port 6789 to that host. There are three possible outcomes:

- *The source host receives a TCP SYNACK segment from the target host.* Since this means that an application is running with TCP port 6789 on the target post, nmap returns "open."
- *The source host receives a TCP RST segment from the target host.* This means that the SYN segment reached the target host, but the target host is not running an application with TCP port 6789. But the attacker at least knows that the segments destined to the host at port 6789 are not blocked by any firewall on the path between source and target hosts. (Firewalls are discussed in Chapter 8.)
- *The source receives nothing.* This likely means that the SYN segment was blocked by an intervening firewall and never reached the target host.

Nmap is a powerful tool, which can "case the joint" not only for open TCP ports, but also for open UDP ports, for firewalls and their configurations, and even for the versions of applications and operating systems. Most of this is done by manipulating TCP connection-management segments [Skoudis 2006]. You can download nmap from www.nmap.org.

This completes our introduction to error control and flow control in TCP. In Section 3.7 we'll return to TCP and look at TCP congestion control in some depth. Before doing so, however, we first step back and examine congestion-control issues in a broader context.

3.6 Principles of Congestion Control

In the previous sections, we examined both the general principles and specific TCP mechanisms used to provide for a reliable data transfer service in the face of packet loss. We mentioned earlier that, in practice, such loss typically results from the overflowing of router buffers as the network becomes congested. Packet retransmission thus treats a symptom of network congestion (the loss of a specific transport-layer segment) but does not treat the cause of network congestion—too many sources attempting to send data at too high a rate. To treat the cause of network congestion, mechanisms are needed to throttle senders in the face of network congestion.

In this section, we consider the problem of congestion control in a general context, seeking to understand why congestion is a bad thing, how network congestion is manifested in the performance received by upper-layer applications, and various approaches that can be taken to avoid, or react to, network congestion. This more general study of congestion control is appropriate since, as with reliable data transfer, it is high on our “top-ten” list of fundamentally important problems in networking. We conclude this section with a discussion of congestion control in the **available bit-rate (ABR)** service in **asynchronous transfer mode (ATM)** networks. The following section contains a detailed study of TCP’s congestion-control algorithm.

3.6.1 The Causes and the Costs of Congestion

Let’s begin our general study of congestion control by examining three increasingly complex scenarios in which congestion occurs. In each case, we’ll look at why congestion occurs in the first place and at the cost of congestion (in terms of resources not fully utilized and poor performance received by the end systems). We’ll not (yet) focus on how to react to, or avoid, congestion but rather focus on the simpler issue of understanding what happens as hosts increase their transmission rate and the network becomes congested.

Scenario 1: Two Senders, a Router with Infinite Buffers

We begin by considering perhaps the simplest congestion scenario possible: Two hosts (A and B) each have a connection that shares a single hop between source and destination, as shown in Figure 3.43.

Let’s assume that the application in Host A is sending data into the connection (for example, passing data to the transport-level protocol via a socket) at an average rate of λ_{in} bytes/sec. These data are original in the sense that each unit of data is sent into the socket only once. The underlying transport-level protocol is a

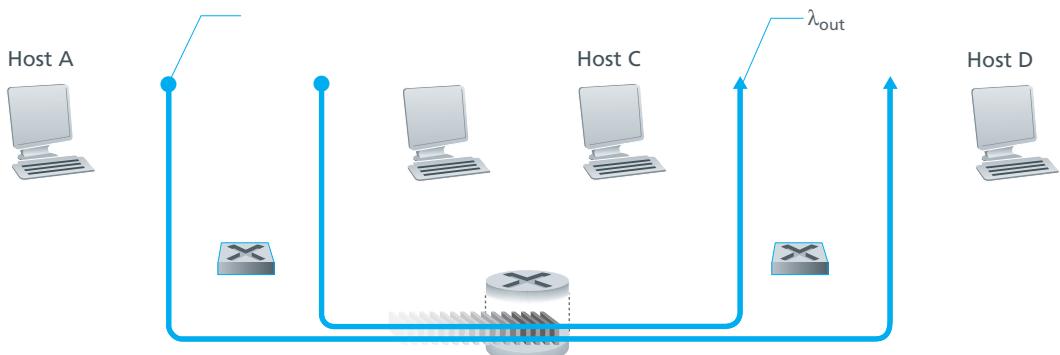


Figure 3.43 ♦ Congestion scenario 1: Two connections sharing a single hop with infinite buffers

simple one. Data is encapsulated and sent; no error recovery (for example, retransmission), flow control, or congestion control is performed. Ignoring the additional overhead due to adding transport- and lower-layer header information, the rate at which Host A offers traffic to the router in this first scenario is thus λ_{in} bytes/sec. Host B operates in a similar manner, and we assume for simplicity that it too is sending at a rate of λ_{in} bytes/sec. Packets from Hosts A and B pass through a router and over a shared outgoing link of capacity R . The router has buffers that allow it to store incoming packets when the packet-arrival rate exceeds the outgoing link's capacity. In this first scenario, we assume that the router has an infinite amount of buffer space.

Figure 3.44 plots the performance of Host A's connection under this first scenario. The left graph plots the **per-connection throughput** (number of bytes per second at the receiver) as a function of the connection-sending rate. For a sending rate between 0 and $R/2$, the throughput at the receiver equals the sender's sending rate—everything sent by the sender is received at the receiver with a finite delay. When the sending rate is above $R/2$, however, the throughput is only $R/2$. This upper limit on throughput is a consequence of the sharing of link capacity between two connections. The link simply cannot deliver packets to a receiver at a steady-state rate that exceeds $R/2$. No matter how high Hosts A and B set their sending rates, they will each never see a throughput higher than $R/2$.

Achieving a per-connection throughput of $R/2$ might actually appear to be a good thing, because the link is fully utilized in delivering packets to their destinations. The right-hand graph in Figure 3.44, however, shows the consequence of operating near link capacity. As the sending rate approaches $R/2$ (from the left), the average delay becomes larger and larger. When the sending rate exceeds $R/2$, the

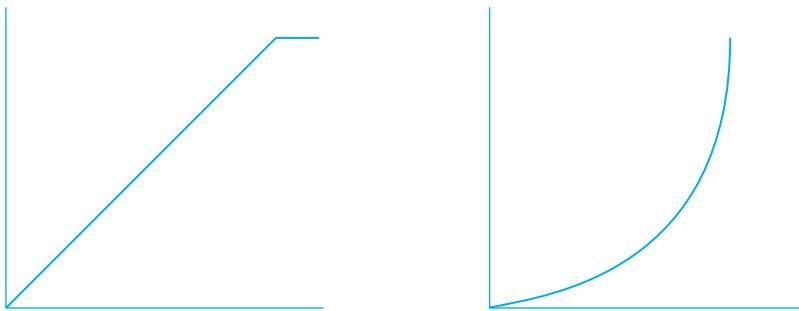


Figure 3.44 ♦ Congestion scenario 1: Throughput and delay as a function of host sending rate

average number of queued packets in the router is unbounded, and the average delay between source and destination becomes infinite (assuming that the connections operate at these sending rates for an infinite period of time and there is an infinite amount of buffering available). Thus, while operating at an aggregate throughput of near R may be ideal from a throughput standpoint, it is far from ideal from a delay standpoint. *Even in this (extremely) idealized scenario, we've already found one cost of a congested network—large queuing delays are experienced as the packet-arrival rate nears the link capacity.*

Scenario 2: Two Senders and a Router with Finite Buffers

Let us now slightly modify scenario 1 in the following two ways (see Figure 3.45). First, the amount of router buffering is assumed to be finite. A consequence of this real-world assumption is that packets will be dropped when arriving to an already-full buffer. Second, we assume that each connection is reliable. If a packet containing a transport-level segment is dropped at the router, the sender will eventually retransmit it. Because packets can be retransmitted, we must now be more careful with our use of the term *sending rate*. Specifically, let us again denote the rate at which the application sends original data into the socket by λ_{in} bytes/sec. The rate at which the transport layer sends segments (containing original data *and* retransmitted data) into the network will be denoted λ'_{in} bytes/sec. λ'_{in} is sometimes referred to as the **offered load** to the network.

The performance realized under scenario 2 will now depend strongly on how retransmission is performed. First, consider the unrealistic case that Host A is able to somehow (magically!) determine whether or not a buffer is free in the router and thus sends a packet only when a buffer is free. In this case, no loss would occur,

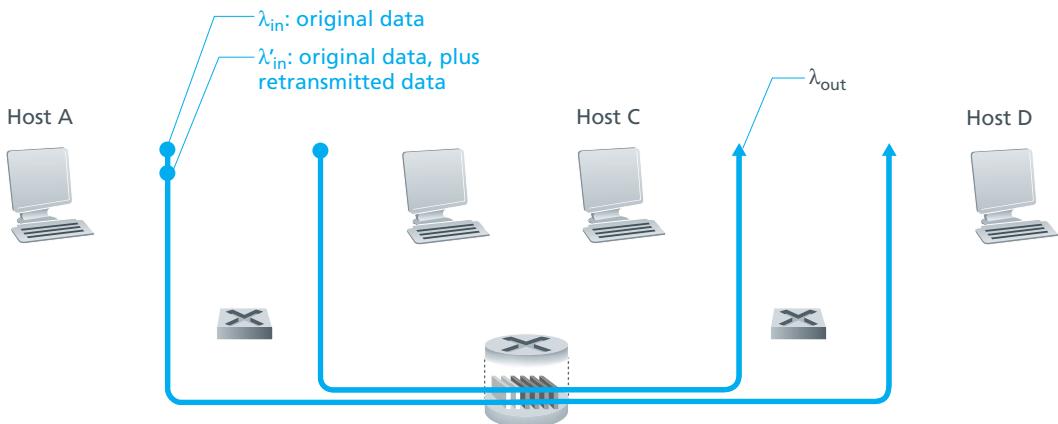


Figure 3.45 ♦ Scenario 2: Two hosts (with retransmissions) and a router with finite buffers

λ_{in} would be equal to λ'_{in} , and the throughput of the connection would be equal to λ_{in} . This case is shown in Figure 3.46(a). From a throughput standpoint, performance is ideal—everything that is sent is received. Note that the average host sending rate cannot exceed $R/2$ under this scenario, since packet loss is assumed never to occur.

Consider next the slightly more realistic case that the sender retransmits only when a packet is known for certain to be lost. (Again, this assumption is a bit of a stretch. However, it is possible that the sending host might set its timeout large enough to be virtually assured that a packet that has not been acknowledged has been lost.) In this case, the performance might look something like that shown in Figure 3.46(b). To appreciate what is happening here, consider the case that the offered load, λ'_{in} (the rate of original data transmission plus retransmissions), equals $R/2$. According to Figure 3.46(b), at this value of the offered load, the rate at which data are delivered to the receiver application is $R/3$. Thus, out of the $0.5R$ units of data transmitted, $0.333R$ bytes/sec (on average) are original data and $0.166R$ bytes/sec (on average) are retransmitted data. *We see here another cost of a congested network—the sender must perform retransmissions in order to compensate for dropped (lost) packets due to buffer overflow.*

Finally, let us consider the case that the sender may time out prematurely and retransmit a packet that has been delayed in the queue but not yet lost. In this case, both the original data packet and the retransmission may reach the receiver. Of

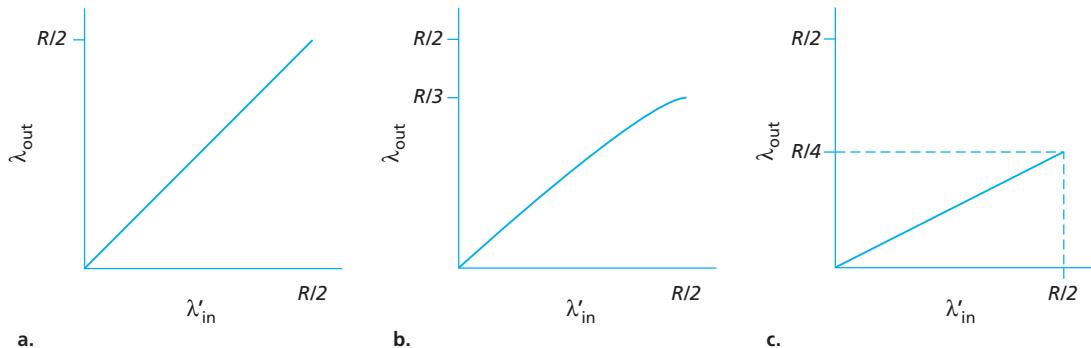


Figure 3.46 ♦ Scenario 2 performance with finite buffers

course, the receiver needs but one copy of this packet and will discard the retransmission. In this case, the work done by the router in forwarding the retransmitted copy of the original packet was wasted, as the receiver will have already received the original copy of this packet. The router would have better used the link transmission capacity to send a different packet instead. *Here then is yet another cost of a congested network—unnecessary retransmissions by the sender in the face of large delays may cause a router to use its link bandwidth to forward unnecessary copies of a packet.* Figure 3.46 (c) shows the throughput versus offered load when each packet is assumed to be forwarded (on average) twice by the router. Since each packet is forwarded twice, the throughput will have an asymptotic value of $R/4$ as the offered load approaches $R/2$.

Scenario 3: Four Senders, Routers with Finite Buffers, and Multihop Paths

In our final congestion scenario, four hosts transmit packets, each over overlapping two-hop paths, as shown in Figure 3.47. We again assume that each host uses a timeout/retransmission mechanism to implement a reliable data transfer service, that all hosts have the same value of λ_{in} , and that all router links have capacity R bytes/sec.

Let's consider the connection from Host A to Host C, passing through routers R1 and R2. The A–C connection shares router R1 with the D–B connection and shares router R2 with the B–D connection. For extremely small values of λ_{in} , buffer overflows are rare (as in congestion scenarios 1 and 2), and the throughput approximately equals the offered load. For slightly larger values of λ_{in} , the corresponding throughput is also larger, since more original data is being transmitted into the

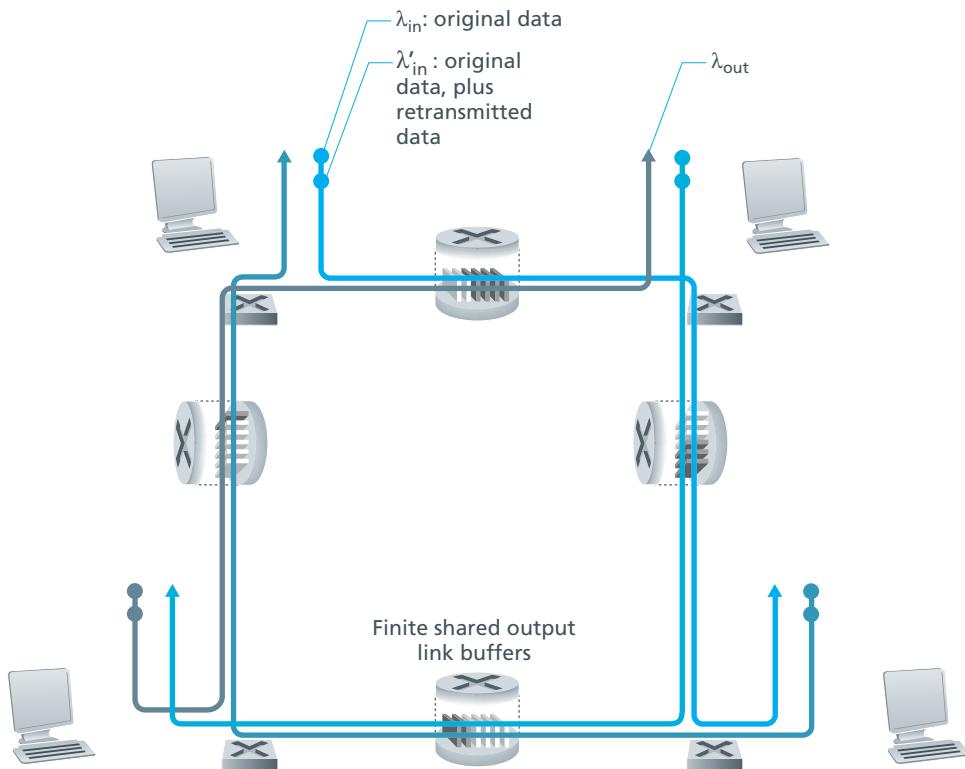


Figure 3.47 ♦ Four senders, routers with finite buffers, and multihop paths

network and delivered to the destination, and overflows are still rare. Thus, for small values of λ_{in} , an increase in λ_{in} results in an increase in λ_{out} .

Having considered the case of extremely low traffic, let's next examine the case that λ_{in} (and hence λ'_{in}) is extremely large. Consider router R2. The A–C traffic arriving to router R2 (which arrives at R2 after being forwarded from R1) can have an arrival rate at R2 that is at most R , the capacity of the link from R1 to R2, regardless of the value of λ_{in} . If λ'_{in} is extremely large for all connections (including the B–D connection), then the arrival rate of B–D traffic at R2 can be much larger than that of the A–C traffic. Because the A–C and B–D traffic must compete at router R2 for the limited amount of buffer space, the amount of A–C traffic that successfully gets through R2 (that is, is not lost due to buffer overflow) becomes smaller and smaller as the offered load from B–D gets larger and larger. In the limit, as the offered load approaches infinity, an empty buffer at R2

is immediately filled by a B–D packet, and the throughput of the A–C connection at R2 goes to zero. This, in turn, *implies that the A–C end-to-end throughput goes to zero* in the limit of heavy traffic. These considerations give rise to the offered load versus throughput tradeoff shown in Figure 3.48.

The reason for the eventual decrease in throughput with increasing offered load is evident when one considers the amount of wasted work done by the network. In the high-traffic scenario outlined above, whenever a packet is dropped at a second-hop router, the work done by the first-hop router in forwarding a packet to the second-hop router ends up being “wasted.” The network would have been equally well off (more accurately, equally bad off) if the first router had simply discarded that packet and remained idle. More to the point, the transmission capacity used at the first router to forward the packet to the second router could have been much more profitably used to transmit a different packet. (For example, when selecting a packet for transmission, it might be better for a router to give priority to packets that have already traversed some number of upstream routers.) *So here we see yet another cost of dropping a packet due to congestion—when a packet is dropped along a path, the transmission capacity that was used at each of the upstream links to forward that packet to the point at which it is dropped ends up having been wasted.*

3.6.2 Approaches to Congestion Control

In Section 3.7, we’ll examine TCP’s specific approach to congestion control in great detail. Here, we identify the two broad approaches to congestion control that are taken in practice and discuss specific network architectures and congestion-control protocols embodying these approaches.

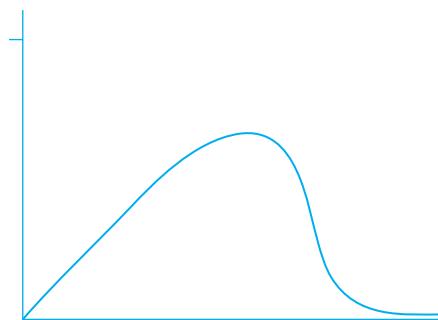


Figure 3.48 ♦ Scenario 3 performance with finite buffers and multihop paths

At the broadest level, we can distinguish among congestion-control approaches by whether the network layer provides any explicit assistance to the transport layer for congestion-control purposes:

- *End-to-end congestion control.* In an end-to-end approach to congestion control, the network layer provides *no explicit support* to the transport layer for congestion-control purposes. Even the presence of congestion in the network must be inferred by the end systems based only on observed network behavior (for example, packet loss and delay). We will see in Section 3.7 that TCP must necessarily take this end-to-end approach toward congestion control, since the IP layer provides no feedback to the end systems regarding network congestion. TCP segment loss (as indicated by a timeout or a triple duplicate acknowledgment) is taken as an indication of network congestion and TCP decreases its window size accordingly. We will also see a more recent proposal for TCP congestion control that uses increasing round-trip delay values as indicators of increased network congestion.
- *Network-assisted congestion control.* With network-assisted congestion control, network-layer components (that is, routers) provide explicit feedback to the sender regarding the congestion state in the network. This feedback may be as simple as a single bit indicating congestion at a link. This approach was taken in the early IBM SNA [Schwartz 1982] and DEC DECnet [Jain 1989; Ramakrishnan 1990] architectures, was recently proposed for TCP/IP networks [Floyd TCP 1994; RFC 3168], and is used in ATM available bit-rate (ABR) congestion control as well, as discussed below. More sophisticated network feedback is also possible. For example, one form of ATM ABR congestion control that we will study shortly allows a router to inform the sender explicitly of the transmission rate it (the router) can support on an outgoing link. The XCP protocol [Katabi 2002] provides router-computed feedback to each source, carried in the packet header, regarding how that source should increase or decrease its transmission rate.

For network-assisted congestion control, congestion information is typically fed back from the network to the sender in one of two ways, as shown in Figure 3.49. Direct feedback may be sent from a network router to the sender. This form of notification typically takes the form of a **choke packet** (essentially saying, “I’m congested!”). The second form of notification occurs when a router marks/updates a field in a packet flowing from sender to receiver to indicate congestion. Upon receipt of a marked packet, the receiver then notifies the sender of the congestion indication. Note that this latter form of notification takes at least a full round-trip time.

3.6.3 Network-Assisted Congestion-Control Example: ATM ABR Congestion Control

We conclude this section with a brief case study of the congestion-control algorithm in ATM ABR—a protocol that takes a network-assisted approach toward congestion control. We stress that our goal here is not to describe aspects of the ATM architecture

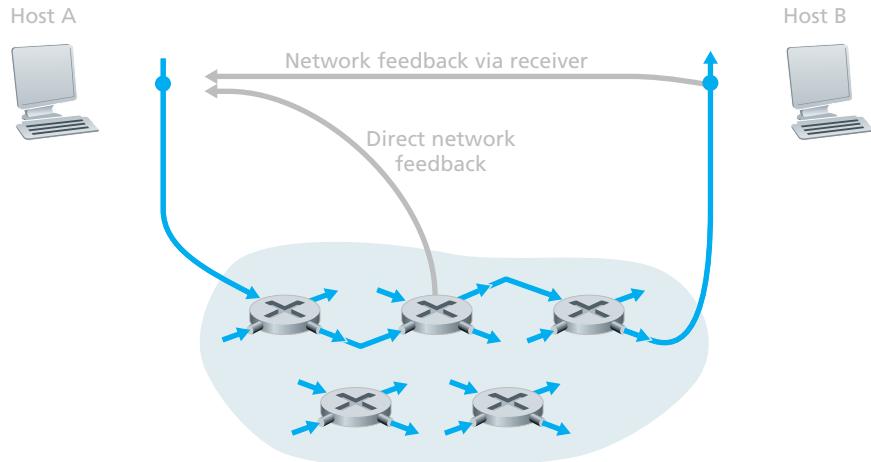


Figure 3.49 ♦ Two feedback pathways for network-indicated congestion information

in great detail, but rather to illustrate a protocol that takes a markedly different approach toward congestion control from that of the Internet's TCP protocol. Indeed, we only present below those few aspects of the ATM architecture that are needed to understand ABR congestion control.

Fundamentally ATM takes a virtual-circuit (VC) oriented approach toward packet switching. Recall from our discussion in Chapter 1, this means that each switch on the source-to-destination path will maintain state about the source-to-destination VC. This per-VC state allows a switch to track the behavior of individual senders (e.g., tracking their average transmission rate) and to take source-specific congestion-control actions (such as explicitly signaling to the sender to reduce its rate when the switch becomes congested). This per-VC state at network switches makes ATM ideally suited to perform network-assisted congestion control.

ABR has been designed as an elastic data transfer service in a manner reminiscent of TCP. When the network is underloaded, ABR service should be able to take advantage of the spare available bandwidth; when the network is congested, ABR service should throttle its transmission rate to some predetermined minimum transmission rate. A detailed tutorial on ATM ABR congestion control and traffic management is provided in [Jain 1996].

Figure 3.50 shows the framework for ATM ABR congestion control. In our discussion we adopt ATM terminology (for example, using the term *switch* rather than *router*, and the term *cell* rather than *packet*). With ATM ABR service, data cells are transmitted from a source to a destination through a series of intermediate switches. Interspersed with the data cells are **resource-management cells**

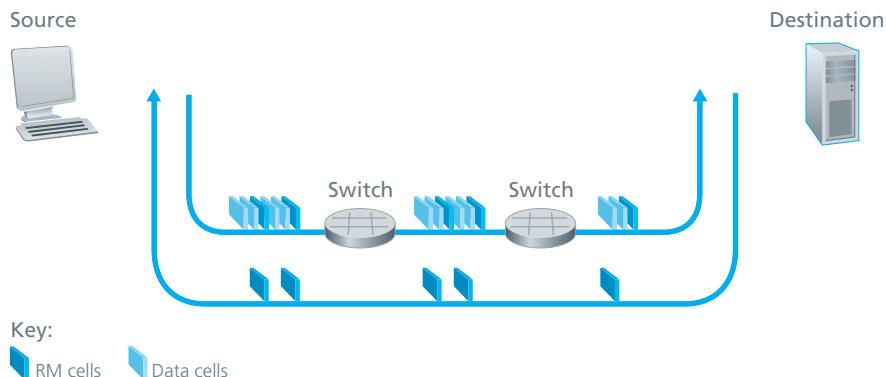


Figure 3.50 ♦ Congestion-control framework for ATM ABR service

(RM cells); these RM cells can be used to convey congestion-related information among the hosts and switches. When an RM cell arrives at a destination, it will be turned around and sent back to the sender (possibly after the destination has modified the contents of the RM cell). It is also possible for a switch to generate an RM cell itself and send this RM cell directly to a source. RM cells can thus be used to provide both direct network feedback and network feedback via the receiver, as shown in Figure 3.50.

ATM ABR congestion control is a rate-based approach. That is, the sender explicitly computes a maximum rate at which it can send and regulates itself accordingly. ABR provides three mechanisms for signaling congestion-related information from the switches to the receiver:

- *EFCI bit.* Each *data cell* contains an **explicit forward congestion indication (EFCI) bit**. A congested network switch can set the EFCI bit in a data cell to 1 to signal congestion to the destination host. The destination must check the EFCI bit in all received data cells. When an RM cell arrives at the destination, if the most recently received data cell had the EFCI bit set to 1, then the destination sets the congestion indication bit (the CI bit) of the RM cell to 1 and sends the RM cell back to the sender. Using the EFCI in data cells and the CI bit in RM cells, a sender can thus be notified about congestion at a network switch.
- *CI and NI bits.* As noted above, sender-to-receiver RM cells are interspersed with data cells. The rate of RM cell interspersion is a tunable parameter, with the default value being one RM cell every 32 data cells. These RM cells have a **congestion indication (CI) bit** and a **no increase (NI) bit** that can be set by a

congested network switch. Specifically, a switch can set the NI bit in a passing RM cell to 1 under mild congestion and can set the CI bit to 1 under severe congestion conditions. When a destination host receives an RM cell, it will send the RM cell back to the sender with its CI and NI bits intact (except that CI may be set to 1 by the destination as a result of the EFCI mechanism described above).

- *ER setting.* Each RM cell also contains a 2-byte **explicit rate (ER) field**. A congested switch may lower the value contained in the ER field in a passing RM cell. In this manner, the ER field will be set to the minimum supportable rate of all switches on the source-to-destination path.

An ATM ABR source adjusts the rate at which it can send cells as a function of the CI, NI, and ER values in a returned RM cell. The rules for making this rate adjustment are rather complicated and a bit tedious. The interested reader is referred to [Jain 1996] for details.

3.7 TCP Congestion Control

In this section we return to our study of TCP. As we learned in Section 3.5, TCP provides a reliable transport service between two processes running on different hosts. Another key component of TCP is its congestion-control mechanism. As indicated in the previous section, TCP must use end-to-end congestion control rather than network-assisted congestion control, since the IP layer provides no explicit feedback to the end systems regarding network congestion.

The approach taken by TCP is to have each sender limit the rate at which it sends traffic into its connection as a function of perceived network congestion. If a TCP sender perceives that there is little congestion on the path between itself and the destination, then the TCP sender increases its send rate; if the sender perceives that there is congestion along the path, then the sender reduces its send rate. But this approach raises three questions. First, how does a TCP sender limit the rate at which it sends traffic into its connection? Second, how does a TCP sender perceive that there is congestion on the path between itself and the destination? And third, what algorithm should the sender use to change its send rate as a function of perceived end-to-end congestion?

Let's first examine how a TCP sender limits the rate at which it sends traffic into its connection. In Section 3.5 we saw that each side of a TCP connection consists of a receive buffer, a send buffer, and several variables (`LastByteRead`, `rwnd`, and so on). The TCP congestion-control mechanism operating at the sender keeps track of an additional variable, the **congestion window**. The congestion window, denoted `cwnd`, imposes a constraint on the rate at which a TCP sender can send traffic

into the network. Specifically, the amount of unacknowledged data at a sender may not exceed the minimum of `cwnd` and `rwnd`, that is:

$$\text{LastByteSent} - \text{LastByteAcked} \leq \min\{\text{cwnd}, \text{rwnd}\}$$

In order to focus on congestion control (as opposed to flow control), let us henceforth assume that the TCP receive buffer is so large that the receive-window constraint can be ignored; thus, the amount of unacknowledged data at the sender is solely limited by `cwnd`. We will also assume that the sender always has data to send, i.e., that all segments in the congestion window are sent.

The constraint above limits the amount of unacknowledged data at the sender and therefore indirectly limits the sender's send rate. To see this, consider a connection for which loss and packet transmission delays are negligible. Then, roughly, at the beginning of every RTT, the constraint permits the sender to send `cwnd` bytes of data into the connection; at the end of the RTT the sender receives acknowledgments for the data. *Thus the sender's send rate is roughly cwnd/RTT bytes/sec. By adjusting the value of cwnd, the sender can therefore adjust the rate at which it sends data into its connection.*

Let's next consider how a TCP sender perceives that there is congestion on the path between itself and the destination. Let us define a "loss event" at a TCP sender as the occurrence of either a timeout or the receipt of three duplicate ACKs from the receiver. (Recall our discussion in Section 3.5.4 of the timeout event in Figure 3.33 and the subsequent modification to include fast retransmit on receipt of three duplicate ACKs.) When there is excessive congestion, then one (or more) router buffers along the path overflows, causing a datagram (containing a TCP segment) to be dropped. The dropped datagram, in turn, results in a loss event at the sender—either a timeout or the receipt of three duplicate ACKs—which is taken by the sender to be an indication of congestion on the sender-to-receiver path.

Having considered how congestion is detected, let's next consider the more optimistic case when the network is congestion-free, that is, when a loss event doesn't occur. In this case, acknowledgments for previously unacknowledged segments will be received at the TCP sender. As we'll see, TCP will take the arrival of these acknowledgments as an indication that all is well—that segments being transmitted into the network are being successfully delivered to the destination—and will use acknowledgments to increase its congestion window size (and hence its transmission rate). Note that if acknowledgments arrive at a relatively slow rate (e.g., if the end-end path has high delay or contains a low-bandwidth link), then the congestion window will be increased at a relatively slow rate. On the other hand, if acknowledgments arrive at a high rate, then the congestion window will be increased more quickly. Because TCP uses

acknowledgments to trigger (or clock) its increase in congestion window size, TCP is said to be **self-clocking**.

Given the mechanism of adjusting the value of $cwnd$ to control the sending rate, the critical question remains: *How* should a TCP sender determine the rate at which it should send? If TCP senders collectively send too fast, they can congest the network, leading to the type of congestion collapse that we saw in Figure 3.48. Indeed, the version of TCP that we'll study shortly was developed in response to observed Internet congestion collapse [Jacobson 1988] under earlier versions of TCP. However, if TCP senders are too cautious and send too slowly, they could under utilize the bandwidth in the network; that is, the TCP senders could send at a higher rate without congesting the network. How then do the TCP senders determine their sending rates such that they don't congest the network but at the same time make use of all the available bandwidth? Are TCP senders explicitly coordinated, or is there a distributed approach in which the TCP senders can set their sending rates based only on local information? TCP answers these questions using the following guiding principles:

- *A lost segment implies congestion, and hence, the TCP sender's rate should be decreased when a segment is lost.* Recall from our discussion in Section 3.5.4, that a timeout event or the receipt of four acknowledgments for a given segment (one original ACK and then three duplicate ACKs) is interpreted as an implicit “loss event” indication of the segment following the quadruply ACKed segment, triggering a retransmission of the lost segment. From a congestion-control standpoint, the question is how the TCP sender should decrease its congestion window size, and hence its sending rate, in response to this inferred loss event.
- *An acknowledged segment indicates that the network is delivering the sender's segments to the receiver, and hence, the sender's rate can be increased when an ACK arrives for a previously unacknowledged segment.* The arrival of acknowledgments is taken as an implicit indication that all is well—segments are being successfully delivered from sender to receiver, and the network is thus not congested. The congestion window size can thus be increased.
- *Bandwidth probing.* Given ACKs indicating a congestion-free source-to-destination path and loss events indicating a congested path, TCP's strategy for adjusting its transmission rate is to increase its rate in response to arriving ACKs until a loss event occurs, at which point, the transmission rate is decreased. The TCP sender thus increases its transmission rate to probe for the rate that at which congestion onset begins, backs off from that rate, and then to begins probing again to see if the congestion onset rate has changed. The TCP sender's behavior is perhaps analogous to the child who requests (and gets) more and more goodies until finally he/she is finally told “No!”, backs off a bit, but then begins making requests

again shortly afterwards. Note that there is no explicit signaling of congestion state by the network—ACKs and loss events serve as implicit signals—and that each TCP sender acts on local information asynchronously from other TCP senders.

Given this overview of TCP congestion control, we’re now in a position to consider the details of the celebrated **TCP congestion-control algorithm**, which was first described in [Jacobson 1988] and is standardized in [RFC 5681]. The algorithm has three major components: (1) slow start, (2) congestion avoidance, and (3) fast recovery. Slow start and congestion avoidance are mandatory components of TCP, differing in how they increase the size of `cwnd` in response to received ACKs. We’ll see shortly that slow start increases the size of `cwnd` more rapidly (despite its name!) than congestion avoidance. Fast recovery is recommended, but not required, for TCP senders.

Slow Start

When a TCP connection begins, the value of `cwnd` is typically initialized to a small value of 1 MSS [RFC 3390], resulting in an initial sending rate of roughly MSS/RTT . For example, if $\text{MSS} = 500$ bytes and $\text{RTT} = 200$ msec, the resulting initial sending rate is only about 20 kbps. Since the available bandwidth to the TCP sender may be much larger than MSS/RTT , the TCP sender would like to find the amount of available bandwidth quickly. Thus, in the **slow-start** state, the value of `cwnd` begins at 1 MSS and increases by 1 MSS every time a transmitted segment is first acknowledged. In the example of Figure 3.51, TCP sends the first segment into the network and waits for an acknowledgment. When this acknowledgment arrives, the TCP sender increases the congestion window by one MSS and sends out two maximum-sized segments. These segments are then acknowledged, with the sender increasing the congestion window by 1 MSS for each of the acknowledged segments, giving a congestion window of 4 MSS, and so on. This process results in a doubling of the sending rate every RTT. Thus, the TCP send rate starts slow but grows exponentially during the slow start phase.

But when should this exponential growth end? Slow start provides several answers to this question. First, if there is a loss event (i.e., congestion) indicated by a timeout, the TCP sender sets the value of `cwnd` to 1 and begins the slow start process anew. It also sets the value of a second state variable, `ssthresh` (shorthand for “slow start threshold”) to $\text{cwnd}/2$ —half of the value of the congestion window value when congestion was detected. The second way in which slow start may end is directly tied to the value of `ssthresh`. Since `ssthresh` is half the value of `cwnd` when congestion was last detected, it might be a bit reckless to keep doubling `cwnd` when it reaches or surpasses the value of `ssthresh`. Thus, when the value of `cwnd` equals `ssthresh`, slow start ends and TCP transitions into congestion avoidance mode. As we’ll see, TCP increases



PRINCIPLES IN PRACTICE

TCP SPLITTING: OPTIMIZING THE PERFORMANCE OF CLOUD SERVICES

For cloud services such as search, e-mail, and social networks, it is desirable to provide a high-level of responsiveness, ideally giving users the illusion that the services are running within their own end systems (including their smartphones). This can be a major challenge, as users are often located far away from the data centers that are responsible for serving the dynamic content associated with the cloud services. Indeed, if the end system is far from a data center, then the RTT will be large, potentially leading to poor response time performance due to TCP slow start.

As a case study, consider the delay in receiving a response for a search query. Typically, the server requires three TCP windows during slow start to deliver the response [Pathak 2010]. Thus the time from when an end system initiates a TCP connection until the time when it receives the last packet of the response is roughly $4 \cdot \text{RTT}$ (one RTT to set up the TCP connection plus three RTTs for the three windows of data) plus the processing time in the data center. These RTT delays can lead to a noticeable delay in returning search results for a significant fraction of queries. Moreover, there can be significant packet loss in access networks, leading to TCP retransmissions and even larger delays.

One way to mitigate this problem and improve user-perceived performance is to (1) deploy front-end servers closer to the users, and (2) utilize **TCP splitting** by breaking the TCP connection at the front-end server. With TCP splitting, the client establishes a TCP connection to the nearby front-end, and the front-end maintains a persistent TCP connection to the data center with a very large TCP congestion window [Tariq 2008, Pathak 2010, Chen 2011]. With this approach, the response time roughly becomes $4 \cdot \text{RTT}_{\text{FE}} + \text{RTT}_{\text{BE}} + \text{processing time}$, where RTT_{FE} is the round-trip time between client and front-end server, and RTT_{BE} is the round-trip time between the front-end server and the data center (back-end server). If the front-end server is close to client, then this response time approximately becomes RTT plus processing time, since RTT_{FE} is negligibly small and RTT_{BE} is approximately RTT. In summary, TCP splitting can reduce the networking delay roughly from $4 \cdot \text{RTT}$ to RTT, significantly improving user-perceived performance, particularly for users who are far from the nearest data center. TCP splitting also helps reduce TCP retransmission delays caused by losses in access networks. Today, Google and Akamai make extensive use of their CDN servers in access networks (see Section 7.2) to perform TCP splitting for the cloud services they support [Chen 2011].

cwnd more cautiously when in congestion-avoidance mode. The final way in which slow start can end is if three duplicate ACKs are detected, in which case TCP performs a fast retransmit (see Section 3.5.4) and enters the fast recovery state, as discussed below. TCP's behavior in slow start is summarized in the FSM

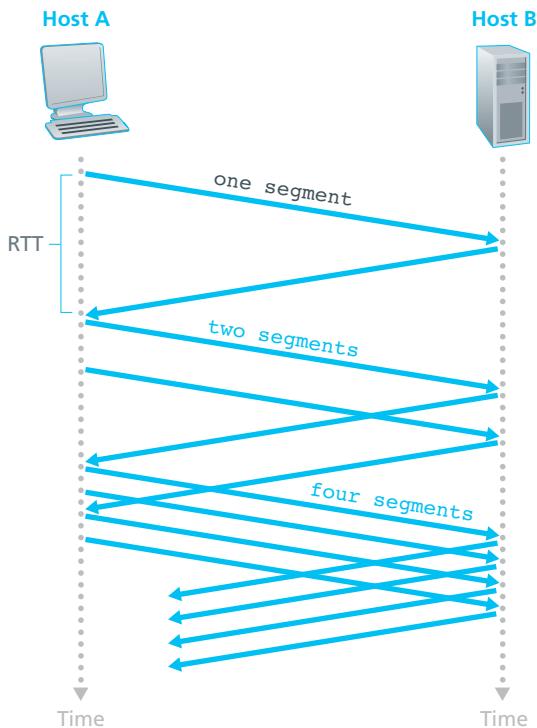


Figure 3.51 ♦ TCP slow start

description of TCP congestion control in Figure 3.52. The slow-start algorithm traces its roots to [Jacobson 1988]; an approach similar to slow start was also proposed independently in [Jain 1986].

Congestion Avoidance

On entry to the congestion-avoidance state, the value of $cwnd$ is approximately half its value when congestion was last encountered—congestion could be just around the corner! Thus, rather than doubling the value of $cwnd$ every RTT, TCP adopts a more conservative approach and increases the value of $cwnd$ by just a single MSS every RTT [RFC 5681]. This can be accomplished in several ways. A common approach is for the TCP sender to increase $cwnd$ by MSS bytes ($MSS/cwnd$) whenever a new acknowledgment arrives. For example, if MSS is 1,460 bytes and $cwnd$ is 14,600 bytes, then 10 segments are being sent within an RTT. Each arriving ACK (assuming one ACK per segment) increases the congestion window size by 1/10

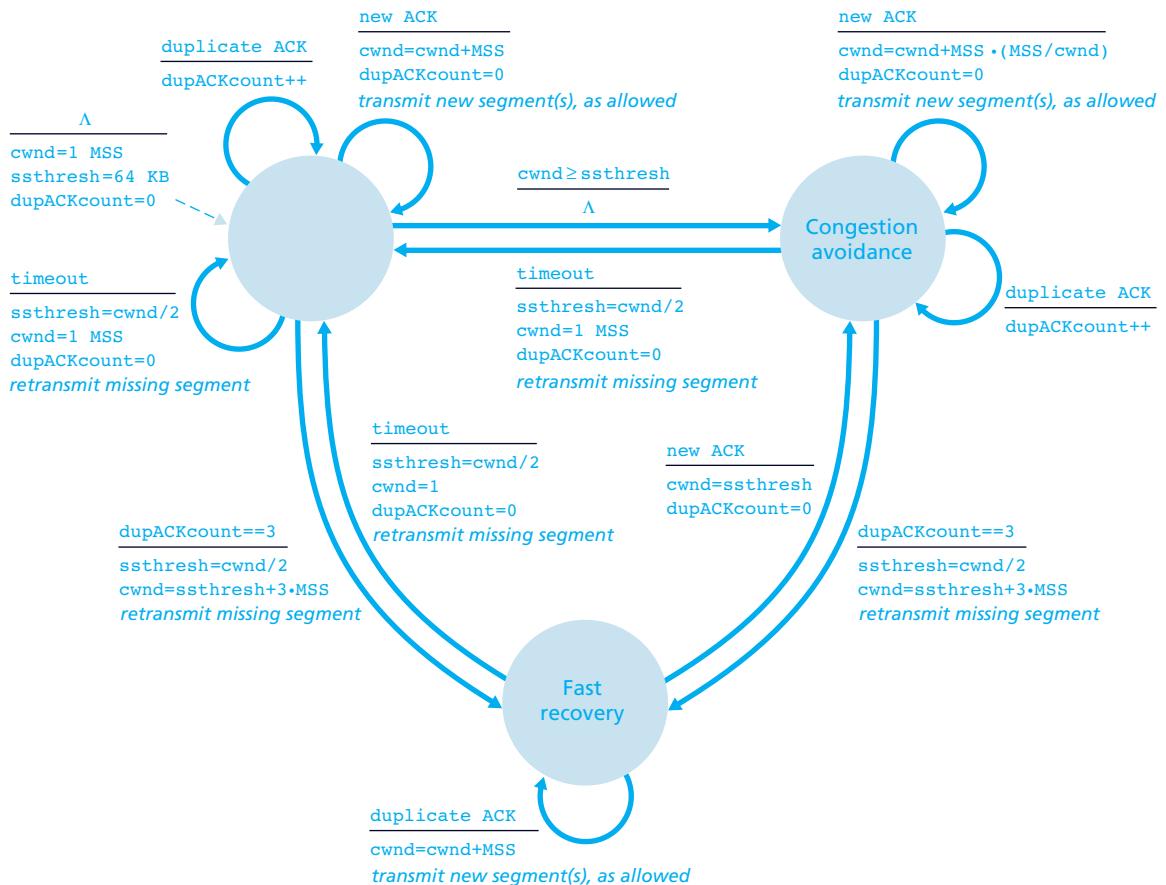


Figure 3.52 ♦ FSM description of TCP congestion control

MSS, and thus, the value of the congestion window will have increased by one MSS after ACKs when all 10 segments have been received.

But when should congestion avoidance's linear increase (of 1 MSS per RTT) end? TCP's congestion-avoidance algorithm behaves the same when a timeout occurs. As in the case of slow start: The value of $cwnd$ is set to 1 MSS, and the value of $ssthresh$ is updated to half the value of $cwnd$ when the loss event occurred. Recall, however, that a loss event also can be triggered by a triple duplicate ACK event. In this case, the network is continuing to deliver segments from sender to receiver (as indicated by the receipt of duplicate ACKs). So TCP's behavior to this type of loss event should be less drastic than with a timeout-indicated loss: TCP halves the value of $cwnd$ (adding in 3 MSS for good measure to account for

the triple duplicate ACKs received) and records the value of `ssthresh` to be half the value of `cwnd` when the triple duplicate ACKs were received. The fast-recovery state is then entered.

Fast Recovery

In fast recovery, the value of `cwnd` is increased by 1 MSS for every duplicate ACK received for the missing segment that caused TCP to enter the fast-recovery state. Eventually, when an ACK arrives for the missing segment, TCP enters the congestion-avoidance state after deflating `cwnd`. If a timeout event occurs, fast recovery transitions to the slow-start state after performing the same actions as in slow start and congestion avoidance: The value of `cwnd` is set to 1 MSS, and the value of `ssthresh` is set to half the value of `cwnd` when the loss event occurred.

Fast recovery is a recommended, but not required, component of TCP [RFC 5681]. It is interesting that an early version of TCP, known as **TCP Tahoe**, unconditionally cut its congestion window to 1 MSS and entered the slow-start phase after either a timeout-indicated or triple-duplicate-ACK-indicated loss event. The newer version of TCP, **TCP Reno**, incorporated fast recovery.



Figure 3.53 illustrates the evolution of TCP's congestion window for both Reno and Tahoe. In this figure, the threshold is initially equal to 8 MSS. For the first eight transmission rounds, Tahoe and Reno take identical actions. The congestion window climbs exponentially fast during slow start and hits the threshold at the fourth round of transmission. The congestion window then climbs linearly until a triple duplicate-ACK event occurs, just after transmission round 8. Note that the congestion window is $12 \cdot MSS$ when this loss event occurs. The value of `ssthresh` is then set to

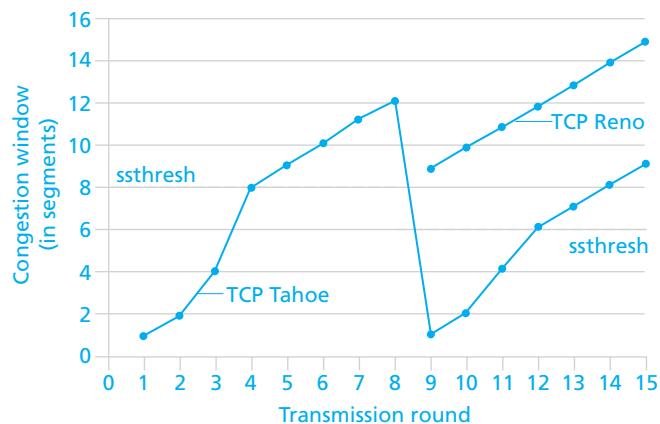


Figure 3.53 ♦ Evolution of TCP's congestion window (Tahoe and Reno)

$0.5 \cdot \text{cwnd} = 6 \cdot \text{MSS}$. Under TCP Reno, the congestion window is set to $\text{cwnd} = 6 \cdot \text{MSS}$ and then grows linearly. Under TCP Tahoe, the congestion window is set to 1 MSS and grows exponentially until it reaches the value of ssthresh , at which point it grows linearly.

Figure 3.52 presents the complete FSM description of TCP’s congestion-control algorithms—slow start, congestion avoidance, and fast recovery. The figure also indicates where transmission of new segments or retransmitted segments can occur. Although it is important to distinguish between TCP error control/retransmission and TCP congestion control, it’s also important to appreciate how these two aspects of TCP are inextricably linked.

TCP Congestion Control: Retrospective

Having delved into the details of slow start, congestion avoidance, and fast recovery, it’s worthwhile to now step back and view the forest from the trees. Ignoring the initial slow-start period when a connection begins and assuming that losses are indicated by triple duplicate ACKs rather than timeouts, TCP’s congestion control consists of linear (additive) increase in cwnd of 1 MSS per RTT and then a halving (multiplicative decrease) of cwnd on a triple duplicate-ACK event. For this reason, TCP congestion control is often referred to as an **additive-increase, multiplicative-decrease (AIMD)** form of congestion control. AIMD congestion control gives rise to the “saw tooth” behavior shown in Figure 3.54, which also nicely illustrates our earlier intuition of TCP “probing” for bandwidth—TCP linearly increases its congestion window size (and hence its transmission rate) until a triple duplicate-ACK event occurs. It then decreases its congestion window size by a factor of two but then again begins increasing it linearly, probing to see if there is additional available bandwidth.

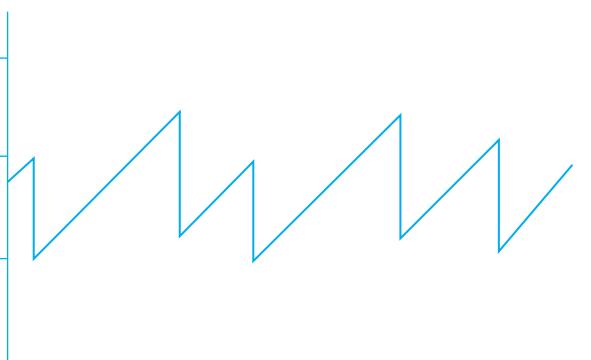


Figure 3.54 ♦ Additive-increase, multiplicative-decrease congestion control

As noted previously, many TCP implementations use the Reno algorithm [Padhye 2001]. Many variations of the Reno algorithm have been proposed [RFC 3782; RFC 2018]. The TCP Vegas algorithm [Brakmo 1995; Ahn 1995] attempts to avoid congestion while maintaining good throughput. The basic idea of Vegas is to (1) detect congestion in the routers between source and destination *before* packet loss occurs, and (2) lower the rate linearly when this imminent packet loss is detected. Imminent packet loss is predicted by observing the RTT. The longer the RTT of the packets, the greater the congestion in the routers. Linux supports a number of congestion-control algorithms (including TCP Reno and TCP Vegas) and allows a system administrator to configure which version of TCP will be used. The default version of TCP in Linux version 2.6.18 was set to CUBIC [Ha 2008], a version of TCP developed for high-bandwidth applications. For a recent survey of the many flavors of TCP, see [Afanasyev 2010].

TCP's AIMD algorithm was developed based on a tremendous amount of engineering insight and experimentation with congestion control in operational networks. Ten years after TCP's development, theoretical analyses showed that TCP's congestion-control algorithm serves as a distributed asynchronous-optimization algorithm that results in several important aspects of user and network performance being simultaneously optimized [Kelly 1998]. A rich theory of congestion control has since been developed [Srikant 2004].

Macroscopic Description of TCP Throughput

Given the saw-toothed behavior of TCP, it's natural to consider what the average throughput (that is, the average rate) of a long-lived TCP connection might be. In this analysis we'll ignore the slow-start phases that occur after timeout events. (These phases are typically very short, since the sender grows out of the phase exponentially fast.) During a particular round-trip interval, the rate at which TCP sends data is a function of the congestion window and the current *RTT*. When the window size is w bytes and the current round-trip time is RTT seconds, then TCP's transmission rate is roughly w/RTT . TCP then probes for additional bandwidth by increasing w by 1 MSS each RTT until a loss event occurs. Denote by W the value of w when a loss event occurs. Assuming that RTT and W are approximately constant over the duration of the connection, the TCP transmission rate ranges from $W/(2 \cdot RTT)$ to W/RTT .

These assumptions lead to a highly simplified macroscopic model for the steady-state behavior of TCP. The network drops a packet from the connection when the rate increases to W/RTT ; the rate is then cut in half and then increases by MSS/RTT every RTT until it again reaches W/RTT . This process repeats itself over and over again. Because TCP's throughput (that is, rate) increases linearly between the two extreme values, we have

$$\text{average throughput of a connection} = \frac{0.75 \cdot W}{RTT}$$

Using this highly idealized model for the steady-state dynamics of TCP, we can also derive an interesting expression that relates a connection’s loss rate to its available bandwidth [Mahdavi 1997]. This derivation is outlined in the homework problems. A more sophisticated model that has been found empirically to agree with measured data is [Padhye 2000].

TCP Over High-Bandwidth Paths

It is important to realize that TCP congestion control has evolved over the years and indeed continues to evolve. For a summary of current TCP variants and discussion of TCP evolution, see [Floyd 2001, RFC 5681, Afanasyev 2010]. What was good for the Internet when the bulk of the TCP connections carried SMTP, FTP, and Telnet traffic is not necessarily good for today’s HTTP-dominated Internet or for a future Internet with services that are still undreamed of.

The need for continued evolution of TCP can be illustrated by considering the high-speed TCP connections that are needed for grid- and cloud-computing applications. For example, consider a TCP connection with 1,500-byte segments and a 100 ms *RTT*, and suppose we want to send data through this connection at 10 Gbps. Following [RFC 3649], we note that using the TCP throughput formula above, in order to achieve a 10 Gbps throughput, the average congestion window size would need to be 83,333 segments. That’s a *lot* of segments, leading us to be rather concerned that one of these 83,333 in-flight segments might be lost. What would happen in the case of a loss? Or, put another way, what fraction of the transmitted segments could be lost that would allow the TCP congestion-control algorithm specified in Figure 3.52 still to achieve the desired 10 Gbps rate? In the homework questions for this chapter, you are led through the derivation of a formula relating the throughput of a TCP connection as a function of the loss rate (L), the round-trip time (RTT), and the maximum segment size (MSS):

$$\text{average throughput of a connection} = \frac{1.22 \cdot MSS}{RTT \sqrt{L}}$$

Using this formula, we can see that in order to achieve a throughput of 10 Gbps, today’s TCP congestion-control algorithm can only tolerate a segment loss probability of $2 \cdot 10^{-10}$ (or equivalently, one loss event for every 5,000,000,000 segments)—a very low rate. This observation has led a number of researchers to investigate new versions of TCP that are specifically designed for such high-speed environments; see [Jin 2004; RFC 3649; Kelly 2003; Ha 2008] for discussions of these efforts.

3.7.1 Fairness

Consider K TCP connections, each with a different end-to-end path, but all passing through a bottleneck link with transmission rate R bps. (By *bottleneck link*, we mean

that for each connection, all the other links along the connection's path are not congested and have abundant transmission capacity as compared with the transmission capacity of the bottleneck link.) Suppose each connection is transferring a large file and there is no UDP traffic passing through the bottleneck link. A congestion-control mechanism is said to be *fair* if the average transmission rate of each connection is approximately R/K ; that is, each connection gets an equal share of the link bandwidth.

Is TCP's AIMD algorithm fair, particularly given that different TCP connections may start at different times and thus may have different window sizes at a given point in time? [Chiu 1989] provides an elegant and intuitive explanation of why TCP congestion control converges to provide an equal share of a bottleneck link's bandwidth among competing TCP connections.

Let's consider the simple case of two TCP connections sharing a single link with transmission rate R , as shown in Figure 3.55. Assume that the two connections have the same MSS and RTT (so that if they have the same congestion window size, then they have the same throughput), that they have a large amount of data to send, and that no other TCP connections or UDP datagrams traverse this shared link. Also, ignore the slow-start phase of TCP and assume the TCP connections are operating in CA mode (AIMD) at all times.

Figure 3.56 plots the throughput realized by the two TCP connections. If TCP is to share the link bandwidth equally between the two connections, then the realized throughput should fall along the 45-degree arrow (equal bandwidth share) emanating from the origin. Ideally, the sum of the two throughputs should equal R . (Certainly, each connection receiving an equal, but zero, share of the link capacity is not a desirable situation!) So the goal should be to have the achieved throughputs fall somewhere near the intersection of the equal bandwidth share line and the full bandwidth utilization line in Figure 3.56.

Suppose that the TCP window sizes are such that at a given point in time, connections 1 and 2 realize throughputs indicated by point A in Figure 3.56. Because the amount of link bandwidth jointly consumed by the two connections is less than

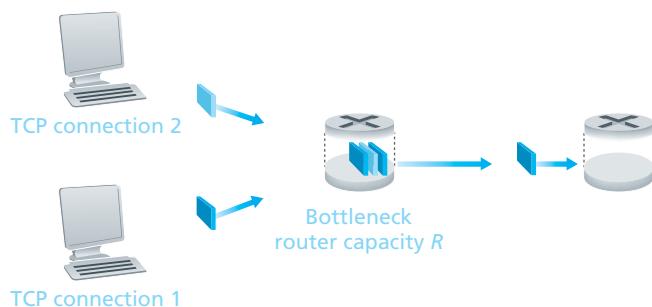


Figure 3.55 ♦ Two TCP connections sharing a single bottleneck link

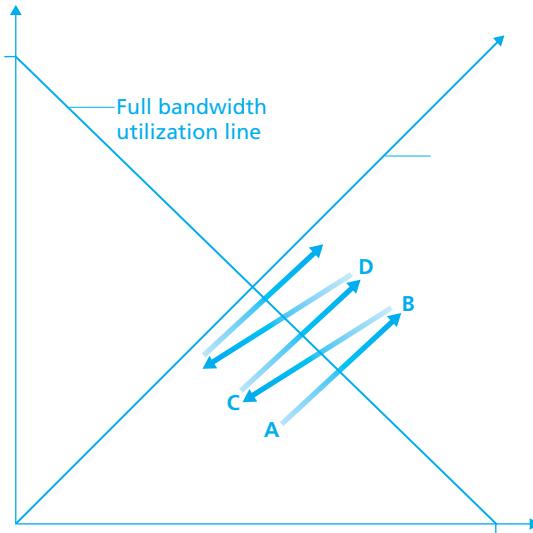


Figure 3.56 ♦ Throughput realized by TCP connections 1 and 2

R , no loss will occur, and both connections will increase their window by 1 MSS per RTT as a result of TCP's congestion-avoidance algorithm. Thus, the joint throughput of the two connections proceeds along a 45-degree line (equal increase for both connections) starting from point A. Eventually, the link bandwidth jointly consumed by the two connections will be greater than R , and eventually packet loss will occur. Suppose that connections 1 and 2 experience packet loss when they realize throughputs indicated by point B. Connections 1 and 2 then decrease their windows by a factor of two. The resulting throughputs realized are thus at point C, halfway along a vector starting at B and ending at the origin. Because the joint bandwidth use is less than R at point C, the two connections again increase their throughputs along a 45-degree line starting from C. Eventually, loss will again occur, for example, at point D, and the two connections again decrease their window sizes by a factor of two, and so on. You should convince yourself that the bandwidth realized by the two connections eventually fluctuates along the equal bandwidth share line. You should also convince yourself that the two connections will converge to this behavior regardless of where they are in the two-dimensional space! Although a number of idealized assumptions lie behind this scenario, it still provides an intuitive feel for why TCP results in an equal sharing of bandwidth among connections.

In our idealized scenario, we assumed that only TCP connections traverse the bottleneck link, that the connections have the same RTT value, and that only a

single TCP connection is associated with a host-destination pair. In practice, these conditions are typically not met, and client-server applications can thus obtain very unequal portions of link bandwidth. In particular, it has been shown that when multiple connections share a common bottleneck, those sessions with a smaller RTT are able to grab the available bandwidth at that link more quickly as it becomes free (that is, open their congestion windows faster) and thus will enjoy higher throughput than those connections with larger RTTs [Lakshman 1997].

Fairness and UDP

We have just seen how TCP congestion control regulates an application’s transmission rate via the congestion window mechanism. Many multimedia applications, such as Internet phone and video conferencing, often do not run over TCP for this very reason—they do not want their transmission rate throttled, even if the network is very congested. Instead, these applications prefer to run over UDP, which does not have built-in congestion control. When running over UDP, applications can pump their audio and video into the network at a constant rate and occasionally lose packets, rather than reduce their rates to “fair” levels at times of congestion and not lose any packets. From the perspective of TCP, the multimedia applications running over UDP are not being fair—they do not cooperate with the other connections nor adjust their transmission rates appropriately. Because TCP congestion control will decrease its transmission rate in the face of increasing congestion (loss), while UDP sources need not, it is possible for UDP sources to crowd out TCP traffic. An area of research today is thus the development of congestion-control mechanisms for the Internet that prevent UDP traffic from bringing the Internet’s throughput to a grinding halt [Floyd 1999; Floyd 2000; Kohler 2006].

Fairness and Parallel TCP Connections

But even if we could force UDP traffic to behave fairly, the fairness problem would still not be completely solved. This is because there is nothing to stop a TCP-based application from using multiple parallel connections. For example, Web browsers often use multiple parallel TCP connections to transfer the multiple objects within a Web page. (The exact number of multiple connections is configurable in most browsers.) When an application uses multiple parallel connections, it gets a larger fraction of the bandwidth in a congested link. As an example, consider a link of rate R supporting nine ongoing client-server applications, with each of the applications using one TCP connection. If a new application comes along and also uses one TCP connection, then each application gets approximately the same transmission rate of $R/10$. But if this new application instead uses 11 parallel TCP connections, then the new application gets an unfair allocation of more than $R/2$. Because Web traffic is so pervasive in the Internet, multiple parallel connections are not uncommon.

3.8 Summary

We began this chapter by studying the services that a transport-layer protocol can provide to network applications. At one extreme, the transport-layer protocol can be very simple and offer a no-frills service to applications, providing only a multiplexing/demultiplexing function for communicating processes. The Internet's UDP protocol is an example of such a no-frills transport-layer protocol. At the other extreme, a transport-layer protocol can provide a variety of guarantees to applications, such as reliable delivery of data, delay guarantees, and bandwidth guarantees. Nevertheless, the services that a transport protocol can provide are often constrained by the service model of the underlying network-layer protocol. If the network-layer protocol cannot provide delay or bandwidth guarantees to transport-layer segments, then the transport-layer protocol cannot provide delay or bandwidth guarantees for the messages sent between processes.

We learned in Section 3.4 that a transport-layer protocol can provide reliable data transfer even if the underlying network layer is unreliable. We saw that providing reliable data transfer has many subtle points, but that the task can be accomplished by carefully combining acknowledgments, timers, retransmissions, and sequence numbers.

Although we covered reliable data transfer in this chapter, we should keep in mind that reliable data transfer can be provided by link-, network-, transport-, or application-layer protocols. Any of the upper four layers of the protocol stack can implement acknowledgments, timers, retransmissions, and sequence numbers and provide reliable data transfer to the layer above. In fact, over the years, engineers and computer scientists have independently designed and implemented link-, network-, transport-, and application-layer protocols that provide reliable data transfer (although many of these protocols have quietly disappeared).

In Section 3.5, we took a close look at TCP, the Internet's connection-oriented and reliable transport-layer protocol. We learned that TCP is complex, involving connection management, flow control, and round-trip time estimation, as well as reliable data transfer. In fact, TCP is actually more complex than our description—we intentionally did not discuss a variety of TCP patches, fixes, and improvements that are widely implemented in various versions of TCP. All of this complexity, however, is hidden from the network application. If a client on one host wants to send data reliably to a server on another host, it simply opens a TCP socket to the server and pumps data into that socket. The client-server application is blissfully unaware of TCP's complexity.

In Section 3.6, we examined congestion control from a broad perspective, and in Section 3.7, we showed how TCP implements congestion control. We learned that congestion control is imperative for the well-being of the network. Without congestion control, a network can easily become gridlocked, with little or no data being transported end-to-end. In Section 3.7 we learned that TCP implements an end-to-end

congestion-control mechanism that additively increases its transmission rate when the TCP connection’s path is judged to be congestion-free, and multiplicatively decreases its transmission rate when loss occurs. This mechanism also strives to give each TCP connection passing through a congested link an equal share of the link bandwidth. We also examined in some depth the impact of TCP connection establishment and slow start on latency. We observed that in many important scenarios, connection establishment and slow start significantly contribute to end-to-end delay. We emphasize once more that while TCP congestion control has evolved over the years, it remains an area of intensive research and will likely continue to evolve in the upcoming years.

Our discussion of specific Internet transport protocols in this chapter has focused on UDP and TCP—the two “work horses” of the Internet transport layer. However, two decades of experience with these two protocols has identified circumstances in which neither is ideally suited. Researchers have thus been busy developing additional transport-layer protocols, several of which are now IETF proposed standards.

The Datagram Congestion Control Protocol (DCCP) [RFC 4340] provides a low-overhead, message-oriented, UDP-like unreliable service, but with an application-selected form of congestion control that is compatible with TCP. If reliable or semi-reliable data transfer is needed by an application, then this would be performed within the application itself, perhaps using the mechanisms we have studied in Section 3.4. DCCP is envisioned for use in applications such as streaming media (see Chapter 7) that can exploit the tradeoff between timeliness and reliability of data delivery, but that want to be responsive to network congestion.

The Stream Control Transmission Protocol (SCTP) [RFC 4960, RFC 3286] is a reliable, message-oriented protocol that allows several different application-level “streams” to be multiplexed through a single SCTP connection (an approach known as “multi-streaming”). From a reliability standpoint, the different streams within the connection are handled separately, so that packet loss in one stream does not affect the delivery of data in other streams. SCTP also allows data to be transferred over two outgoing paths when a host is connected to two or more networks, optional delivery of out-of-order data, and a number of other features. SCTP’s flow- and congestion-control algorithms are essentially the same as in TCP.

The TCP-Friendly Rate Control (TFRC) protocol [RFC 5348] is a congestion-control protocol rather than a full-fledged transport-layer protocol. It specifies a congestion-control mechanism that could be used in another transport protocol such as DCCP (indeed one of the two application-selectable protocols available in DCCP is TFRC). The goal of TFRC is to smooth out the “saw tooth” behavior (see Figure 3.54) in TCP congestion control, while maintaining a long-term sending rate that is “reasonably” close to that of TCP. With a smoother sending rate than TCP, TFRC is well-suited for multimedia applications such as IP telephony or streaming media where such a smooth rate is important. TFRC is an “equation-based” protocol that uses the measured packet loss rate as input to an equation [Padhye 2000] that estimates what TCP’s throughput would be if a TCP session experiences that loss rate. This rate is then taken as TFRC’s target sending rate.

Only the future will tell whether DCCP, SCTP, or TFRC will see widespread deployment. While these protocols clearly provide enhanced capabilities over TCP and UDP, TCP and UDP have proven themselves “good enough” over the years. Whether “better” wins out over “good enough” will depend on a complex mix of technical, social, and business considerations.

In Chapter 1, we said that a computer network can be partitioned into the “network edge” and the “network core.” The network edge covers everything that happens in the end systems. Having now covered the application layer and the transport layer, our discussion of the network edge is complete. It is time to explore the network core! This journey begins in the next chapter, where we’ll study the network layer, and continues into Chapter 5, where we’ll study the link layer.



Homework Problems and Questions

Chapter 3 Review Questions

SECTIONS 3.1–3.3

R1. Suppose the network layer provides the following service. The network layer in the source host accepts a segment of maximum size 1,200 bytes and a destination host address from the transport layer. The network layer then guarantees to deliver the segment to the transport layer at the destination host.

Suppose many network application processes can be running at the destination host.

- a. Design the simplest possible transport-layer protocol that will get application data to the desired process at the destination host. Assume the operating system in the destination host has assigned a 4-byte port number to each running application process.
- b. Modify this protocol so that it provides a “return address” to the destination process.
- c. In your protocols, does the transport layer “have to do anything” in the core of the computer network?

R2. Consider a planet where everyone belongs to a family of six, every family lives in its own house, each house has a unique address, and each person in a given house has a unique name. Suppose this planet has a mail service that delivers letters from source house to destination house. The mail service requires that (1) the letter be in an envelope, and that (2) the address of the destination house (and nothing more) be clearly written on the envelope. Suppose each family has a delegate family member who collects and distributes letters for the other family members. The letters do not necessarily provide any indication of the recipients of the letters.

- a. Using the solution to Problem R1 above as inspiration, describe a protocol that the delegates can use to deliver letters from a sending family member to a receiving family member.
 - b. In your protocol, does the mail service ever have to open the envelope and examine the letter in order to provide its service?
- R3. Consider a TCP connection between Host A and Host B. Suppose that the TCP segments traveling from Host A to Host B have source port number x and destination port number y . What are the source and destination port numbers for the segments traveling from Host B to Host A?
- R4. Describe why an application developer might choose to run an application over UDP rather than TCP.
- R5. Why is it that voice and video traffic is often sent over TCP rather than UDP in today's Internet? (*Hint:* The answer we are looking for has nothing to do with TCP's congestion-control mechanism.)
- R6. Is it possible for an application to enjoy reliable data transfer even when the application runs over UDP? If so, how?
- R7. Suppose a process in Host C has a UDP socket with port number 6789. Suppose both Host A and Host B each send a UDP segment to Host C with destination port number 6789. Will both of these segments be directed to the same socket at Host C? If so, how will the process at Host C know that these two segments originated from two different hosts?
- R8. Suppose that a Web server runs in Host C on port 80. Suppose this Web server uses persistent connections, and is currently receiving requests from two different Hosts, A and B. Are all of the requests being sent through the same socket at Host C? If they are being passed through different sockets, do both of the sockets have port 80? Discuss and explain.

SECTION 3.4

- R9. In our `rdt` protocols, why did we need to introduce sequence numbers?
- R10. In our `rdt` protocols, why did we need to introduce timers?
- R11. Suppose that the roundtrip delay between sender and receiver is constant and known to the sender. Would a timer still be necessary in protocol `rdt 3.0`, assuming that packets can be lost? Explain.
- R12. Visit the Go-Back-N Java applet at the companion Web site.
 - a. Have the source send five packets, and then pause the animation before any of the five packets reach the destination. Then kill the first packet and resume the animation. Describe what happens.
 - b. Repeat the experiment, but now let the first packet reach the destination and kill the first acknowledgment. Describe again what happens.
 - c. Finally, try sending six packets. What happens?

R13. Repeat R12, but now with the Selective Repeat Java applet. How are Selective Repeat and Go-Back-N different?

SECTION 3.5

R14. True or false?

- a. Host A is sending Host B a large file over a TCP connection. Assume Host B has no data to send Host A. Host B will not send acknowledgments to Host A because Host B cannot piggyback the acknowledgments on data.
- b. The size of the TCP `rwnd` never changes throughout the duration of the connection.
- c. Suppose Host A is sending Host B a large file over a TCP connection. The number of unacknowledged bytes that A sends cannot exceed the size of the receive buffer.
- d. Suppose Host A is sending a large file to Host B over a TCP connection. If the sequence number for a segment of this connection is m , then the sequence number for the subsequent segment will necessarily be $m + 1$.
- e. The TCP segment has a field in its header for `rwnd`.
- f. Suppose that the last `SampleRTT` in a TCP connection is equal to 1 sec. The current value of `TimeoutInterval` for the connection will necessarily be ≥ 1 sec.
- g. Suppose Host A sends one segment with sequence number 38 and 4 bytes of data over a TCP connection to Host B. In this same segment the acknowledgment number is necessarily 42.

R15. Suppose Host A sends two TCP segments back to back to Host B over a TCP connection. The first segment has sequence number 90; the second has sequence number 110.

- a. How much data is in the first segment?
- b. Suppose that the first segment is lost but the second segment arrives at B. In the acknowledgment that Host B sends to Host A, what will be the acknowledgment number?

R16. Consider the Telnet example discussed in Section 3.5. A few seconds after the user types the letter ‘C,’ the user types the letter ‘R.’ After typing the letter ‘R,’ how many segments are sent, and what is put in the sequence number and acknowledgment fields of the segments?

SECTION 3.7

R17. Suppose two TCP connections are present over some bottleneck link of rate R bps. Both connections have a huge file to send (in the same direction over the

bottleneck link). The transmissions of the files start at the same time. What transmission rate would TCP like to give to each of the connections?

R18. True or false? Consider congestion control in TCP. When the timer expires at the sender, the value of `ssthresh` is set to one half of its previous value.

R19. In the discussion of TCP splitting in the sidebar in Section 7.2, it was claimed that the response time with TCP splitting is approximately $4 \cdot RTT_{FE} + RTT_{BE} + \text{processing time}$. Justify this claim.



Problems

P1. Suppose Client A initiates a Telnet session with Server S. At about the same time, Client B also initiates a Telnet session with Server S. Provide possible source and destination port numbers for

- The segments sent from A to S.
- The segments sent from B to S.
- The segments sent from S to A.
- The segments sent from S to B.
- If A and B are different hosts, is it possible that the source port number in the segments from A to S is the same as that from B to S?
- How about if they are the same host?

P2. Consider Figure 3.5. What are the source and destination port values in the segments flowing from the server back to the clients' processes? What are the IP addresses in the network-layer datagrams carrying the transport-layer segments?

P3. UDP and TCP use 1s complement for their checksums. Suppose you have the following three 8-bit bytes: 01010011, 01100110, 01110100. What is the 1s complement of the sum of these 8-bit bytes? (Note that although UDP and TCP use 16-bit words in computing the checksum, for this problem you are being asked to consider 8-bit sums.) Show all work. Why is it that UDP takes the 1s complement of the sum; that is, why not just use the sum? With the 1s complement scheme, how does the receiver detect errors? Is it possible that a 1-bit error will go undetected? How about a 2-bit error?

- P4.
- Suppose you have the following 2 bytes: 01011100 and 01100101. What is the 1s complement of the sum of these 2 bytes?
 - Suppose you have the following 2 bytes: 11011010 and 01100101. What is the 1s complement of the sum of these 2 bytes?
 - For the bytes in part (a), give an example where one bit is flipped in each of the 2 bytes and yet the 1s complement doesn't change.

- P5. Suppose that the UDP receiver computes the Internet checksum for the received UDP segment and finds that it matches the value carried in the checksum field. Can the receiver be absolutely certain that no bit errors have occurred? Explain.
- P6. Consider our motivation for correcting protocol `rdt2.1`. Show that the receiver, shown in Figure 3.57, when operating with the sender shown in Figure 3.11, can lead the sender and receiver to enter into a deadlock state, where each is waiting for an event that will never occur.
- P7. In protocol `rdt3.0`, the ACK packets flowing from the receiver to the sender do not have sequence numbers (although they do have an ACK field that contains the sequence number of the packet they are acknowledging). Why is it that our ACK packets do not require sequence numbers?
- P8. Draw the FSM for the receiver side of protocol `rdt3.0`.
- P9. Give a trace of the operation of protocol `rdt3.0` when data packets and acknowledgment packets are garbled. Your trace should be similar to that used in Figure 3.16.
- P10. Consider a channel that can lose packets but has a maximum delay that is known. Modify protocol `rdt2.1` to include sender timeout and retransmit. Informally argue why your protocol can communicate correctly over this channel.

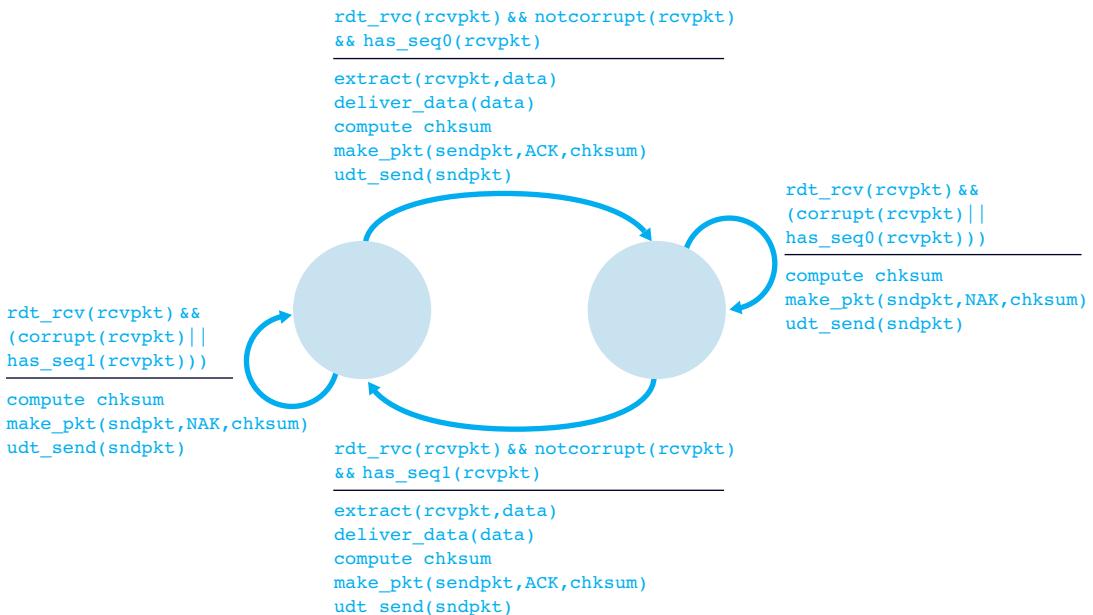


Figure 3.57 ♦ An incorrect receiver for protocol `rdt 2.1`

- P11. Consider the `rdt2.2` receiver in Figure 3.14, and the creation of a new packet in the self-transition (i.e., the transition from the state back to itself) in the Wait-for-0-from-below and the Wait-for-1-from-below states: `sndpkt=make_pkt(ACK, 0, checksum)` and `sndpkt=make_pkt(ACK, 0, checksum)`. Would the protocol work correctly if this action were removed from the self-transition in the Wait-for-1-from-below state? Justify your answer. What if this event were removed from the self-transition in the Wait-for-0-from-below state? [*Hint:* In this latter case, consider what would happen if the first sender-to-receiver packet were corrupted.]
- P12. The sender side of `rdt3.0` simply ignores (that is, takes no action on) all received packets that are either in error or have the wrong value in the `acknum` field of an acknowledgment packet. Suppose that in such circumstances, `rdt3.0` were simply to retransmit the current data packet. Would the protocol still work? (*Hint:* Consider what would happen if there were only bit errors; there are no packet losses but premature timeouts can occur. Consider how many times the n th packet is sent, in the limit as n approaches infinity.)
- P13. Consider the `rdt 3.0` protocol. Draw a diagram showing that if the network connection between the sender and receiver can reorder messages (that is, that two messages propagating in the medium between the sender and receiver can be reordered), then the alternating-bit protocol will not work correctly (make sure you clearly identify the sense in which it will not work correctly). Your diagram should have the sender on the left and the receiver on the right, with the time axis running down the page, showing data (D) and acknowledgment (A) message exchange. Make sure you indicate the sequence number associated with any data or acknowledgment segment.
- P14. Consider a reliable data transfer protocol that uses only negative acknowledgments. Suppose the sender sends data only infrequently. Would a NAK-only protocol be preferable to a protocol that uses ACKs? Why? Now suppose the sender has a lot of data to send and the end-to-end connection experiences few losses. In this second case, would a NAK-only protocol be preferable to a protocol that uses ACKs? Why?
- P15. Consider the cross-country example shown in Figure 3.17. How big would the window size have to be for the channel utilization to be greater than 98 percent? Suppose that the size of a packet is 1,500 bytes, including both header fields and data.
- P16. Suppose an application uses `rdt 3.0` as its transport layer protocol. As the stop-and-wait protocol has very low channel utilization (shown in the cross-country example), the designers of this application let the receiver keep sending back a number (more than two) of alternating ACK 0 and ACK 1 even if

the corresponding data have not arrived at the receiver. Would this application design increase the channel utilization? Why? Are there any potential problems with this approach? Explain.

- P17. Consider two network entities, A and B, which are connected by a perfect bi-directional channel (i.e., any message sent will be received correctly; the channel will not corrupt, lose, or re-order packets). A and B are to deliver data messages to each other in an alternating manner: First, A must deliver a message to B, then B must deliver a message to A, then A must deliver a message to B and so on. If an entity is in a state where it should not attempt to deliver a message to the other side, and there is an event like `rdt_send(data)` call from above that attempts to pass data down for transmission to the other side, this call from above can simply be ignored with a call to `rdt_unable_to_send(data)`, which informs the higher layer that it is currently not able to send data. [Note: This simplifying assumption is made so you don't have to worry about buffering data.]

Draw a FSM specification for this protocol (one FSM for A, and one FSM for B!). Note that you do not have to worry about a reliability mechanism here; the main point of this question is to create a FSM specification that reflects the synchronized behavior of the two entities. You should use the following events and actions that have the same meaning as protocol rdt1.0 in Figure 3.9: `rdt_send(data)`, `packet = make_pkt(data)`, `udt_send(packet)`, `rdt_rcv(packet)`, `extract(packet, data)`, `deliver_data(data)`. Make sure your protocol reflects the strict alternation of sending between A and B. Also, make sure to indicate the initial states for A and B in your FSM descriptions.

- P18. In the generic SR protocol that we studied in Section 3.4.4, the sender transmits a message as soon as it is available (if it is in the window) without waiting for an acknowledgment. Suppose now that we want an SR protocol that sends messages two at a time. That is, the sender will send a pair of messages and will send the next pair of messages only when it knows that both messages in the first pair have been received correctly.

Suppose that the channel may lose messages but will not corrupt or reorder messages. Design an error-control protocol for the unidirectional reliable transfer of messages. Give an FSM description of the sender and receiver. Describe the format of the packets sent between sender and receiver, and vice versa. If you use any procedure calls other than those in Section 3.4 (for example, `udt_send()`, `start_timer()`, `rdt_rcv()`, and so on), clearly state their actions. Give an example (a timeline trace of sender and receiver) showing how your protocol recovers from a lost packet.

- P19. Consider a scenario in which Host A wants to simultaneously send packets to Hosts B and C. A is connected to B and C via a broadcast channel—a packet

sent by A is carried by the channel to both B and C. Suppose that the broadcast channel connecting A, B, and C can independently lose and corrupt packets (and so, for example, a packet sent from A might be correctly received by B, but not by C). Design a stop-and-wait-like error-control protocol for reliably transferring packets from A to B and C, such that A will not get new data from the upper layer until it knows that both B and C have correctly received the current packet. Give FSM descriptions of A and C. (*Hint:* The FSM for B should be essentially the same as for C.) Also, give a description of the packet format(s) used.

- P20. Consider a scenario in which Host A and Host B want to send messages to Host C. Hosts A and C are connected by a channel that can lose and corrupt (but not reorder) messages. Hosts B and C are connected by another channel (independent of the channel connecting A and C) with the same properties. The transport layer at Host C should alternate in delivering messages from A and B to the layer above (that is, it should first deliver the data from a packet from A, then the data from a packet from B, and so on). Design a stop-and-wait-like error-control protocol for reliably transferring packets from A and B to C, with alternating delivery at C as described above. Give FSM descriptions of A and C. (*Hint:* The FSM for B should be essentially the same as for A.) Also, give a description of the packet format(s) used.
- P21. Suppose we have two network entities, A and B. B has a supply of data messages that will be sent to A according to the following conventions. When A gets a request from the layer above to get the next data (D) message from B, A must send a request (R) message to B on the A-to-B channel. Only when B receives an R message can it send a data (D) message back to A on the B-to-A channel. A should deliver exactly one copy of each D message to the layer above. R messages can be lost (but not corrupted) in the A-to-B channel; D messages, once sent, are always delivered correctly. The delay along both channels is unknown and variable.
- Design (give an FSM description of) a protocol that incorporates the appropriate mechanisms to compensate for the loss-prone A-to-B channel and implements message passing to the layer above at entity A, as discussed above. Use only those mechanisms that are absolutely necessary.
- P22. Consider the GBN protocol with a sender window size of 4 and a sequence number range of 1,024. Suppose that at time t , the next in-order packet that the receiver is expecting has a sequence number of k . Assume that the medium does not reorder messages. Answer the following questions:
- What are the possible sets of sequence numbers inside the sender's window at time t ? Justify your answer.
 - What are all possible values of the ACK field in all possible messages currently propagating back to the sender at time t ? Justify your answer.

- P23. Consider the GBN and SR protocols. Suppose the sequence number space is of size k . What is the largest allowable sender window that will avoid the occurrence of problems such as that in Figure 3.27 for each of these protocols?
- P24. Answer true or false to the following questions and briefly justify your answer:
- With the SR protocol, it is possible for the sender to receive an ACK for a packet that falls outside of its current window.
 - With GBN, it is possible for the sender to receive an ACK for a packet that falls outside of its current window.
 - The alternating-bit protocol is the same as the SR protocol with a sender and receiver window size of 1.
 - The alternating-bit protocol is the same as the GBN protocol with a sender and receiver window size of 1.
- P25. We have said that an application may choose UDP for a transport protocol because UDP offers finer application control (than TCP) of what data is sent in a segment and when.
- Why does an application have more control of what data is sent in a segment?
 - Why does an application have more control on when the segment is sent?
- P26. Consider transferring an enormous file of L bytes from Host A to Host B.
Assume an MSS of 536 bytes.
- What is the maximum value of L such that TCP sequence numbers are not exhausted? Recall that the TCP sequence number field has 4 bytes.
 - For the L you obtain in (a), find how long it takes to transmit the file.
Assume that a total of 66 bytes of transport, network, and data-link header are added to each segment before the resulting packet is sent out over a 155 Mbps link. Ignore flow control and congestion control so A can pump out the segments back to back and continuously.
- P27. Host A and B are communicating over a TCP connection, and Host B has already received from A all bytes up through byte 126. Suppose Host A then sends two segments to Host B back-to-back. The first and second segments contain 80 and 40 bytes of data, respectively. In the first segment, the sequence number is 127, the source port number is 302, and the destination port number is 80. Host B sends an acknowledgment whenever it receives a segment from Host A.
- In the second segment sent from Host A to B, what are the sequence number, source port number, and destination port number?
 - If the first segment arrives before the second segment, in the acknowledgment of the first arriving segment, what is the acknowledgment number, the source port number, and the destination port number?

- c. If the second segment arrives before the first segment, in the acknowledgment of the first arriving segment, what is the acknowledgment number?
 - d. Suppose the two segments sent by A arrive in order at B. The first acknowledgment is lost and the second acknowledgment arrives after the first timeout interval. Draw a timing diagram, showing these segments and all other segments and acknowledgments sent. (Assume there is no additional packet loss.) For each segment in your figure, provide the sequence number and the number of bytes of data; for each acknowledgment that you add, provide the acknowledgment number.
- P28. Host A and B are directly connected with a 100 Mbps link. There is one TCP connection between the two hosts, and Host A is sending to Host B an enormous file over this connection. Host A can send its application data into its TCP socket at a rate as high as 120 Mbps but Host B can read out of its TCP receive buffer at a maximum rate of 50 Mbps. Describe the effect of TCP flow control.
- P29. SYN cookies were discussed in Section 3.5.6.
- a. Why is it necessary for the server to use a special initial sequence number in the SYNACK?
 - b. Suppose an attacker knows that a target host uses SYN cookies. Can the attacker create half-open or fully open connections by simply sending an ACK packet to the target? Why or why not?
 - c. Suppose an attacker collects a large amount of initial sequence numbers sent by the server. Can the attacker cause the server to create many fully open connections by sending ACKs with those initial sequence numbers? Why?
- P30. Consider the network shown in Scenario 2 in Section 3.6.1. Suppose both sending hosts A and B have some fixed timeout values.
- a. Argue that increasing the size of the finite buffer of the router might possibly decrease the throughput (λ_{out}).
 - b. Now suppose both hosts dynamically adjust their timeout values (like what TCP does) based on the buffering delay at the router. Would increasing the buffer size help to increase the throughput? Why?
- P31. Suppose that the five measured **SampleRTT** values (see Section 3.5.3) are 106 ms, 120 ms, 140 ms, 90 ms, and 115 ms. Compute the **EstimatedRTT** after each of these SampleRTT values is obtained, using a value of $\alpha = 0.125$ and assuming that the value of **EstimatedRTT** was 100 ms just before the first of these five samples were obtained. Compute also the **DevRTT** after each sample is obtained, assuming a value of $\beta = 0.25$ and assuming the value of **DevRTT** was 5 ms just before the first of these five samples was obtained. Last, compute the TCP **TimeoutInterval** after each of these samples is obtained.

- P32. Consider the TCP procedure for estimating RTT. Suppose that $\alpha = 0.1$. Let `SampleRTT1` be the most recent sample RTT, let `SampleRTT2` be the next most recent sample RTT, and so on.
- For a given TCP connection, suppose four acknowledgments have been returned with corresponding sample RTTs: `SampleRTT4`, `SampleRTT3`, `SampleRTT2`, and `SampleRTT1`. Express `EstimatedRTT` in terms of the four sample RTTs.
 - Generalize your formula for n sample RTTs.
 - For the formula in part (b) let n approach infinity. Comment on why this averaging procedure is called an exponential moving average.
- P33. In Section 3.5.3, we discussed TCP's estimation of RTT. Why do you think TCP avoids measuring the `SampleRTT` for retransmitted segments?
- P34. What is the relationship between the variable `SendBase` in Section 3.5.4 and the variable `LastByteRcvd` in Section 3.5.5?
- P35. What is the relationship between the variable `LastByteRcvd` in Section 3.5.5 and the variable `y` in Section 3.5.4?
- P36. In Section 3.5.4, we saw that TCP waits until it has received three duplicate ACKs before performing a fast retransmit. Why do you think the TCP designers chose not to perform a fast retransmit after the first duplicate ACK for a segment is received?
- P37. Compare GBN, SR, and TCP (no delayed ACK). Assume that the timeout values for all three protocols are sufficiently long such that 5 consecutive data segments and their corresponding ACKs can be received (if not lost in the channel) by the receiving host (Host B) and the sending host (Host A) respectively. Suppose Host A sends 5 data segments to Host B, and the 2nd segment (sent from A) is lost. In the end, all 5 data segments have been correctly received by Host B.
- How many segments has Host A sent in total and how many ACKs has Host B sent in total? What are their sequence numbers? Answer this question for all three protocols.
 - If the timeout values for all three protocol are much longer than 5 RTT, then which protocol successfully delivers all five data segments in shortest time interval?
- P38. In our description of TCP in Figure 3.53, the value of the threshold, `ssthresh`, is set as `ssthresh=cwnd/2` in several places and `ssthresh` value is referred to as being set to half the window size when a loss event occurred. Must the rate at which the sender is sending when the loss event occurred be approximately equal to `cwnd` segments per RTT? Explain your answer. If your answer is no, can you suggest a different manner in which `ssthresh` should be set?

- P39. Consider Figure 3.46(b). If λ'_{in} increases beyond $R/2$, can λ_{out} increase beyond $R/3$? Explain. Now consider Figure 3.46(c). If λ'_{in} increases beyond $R/2$, can λ_{out} increase beyond $R/4$ under the assumption that a packet will be forwarded twice on average from the router to the receiver? Explain.
- P40. Consider Figure 3.58. Assuming TCP Reno is the protocol experiencing the behavior shown above, answer the following questions. In all cases, you should provide a short discussion justifying your answer.
- Identify the intervals of time when TCP slow start is operating.
 - Identify the intervals of time when TCP congestion avoidance is operating.
 - After the 16th transmission round, is segment loss detected by a triple duplicate ACK or by a timeout?
 - After the 22nd transmission round, is segment loss detected by a triple duplicate ACK or by a timeout?
 - What is the initial value of `ssthresh` at the first transmission round?
 - What is the value of `ssthresh` at the 18th transmission round?
 - What is the value of `ssthresh` at the 24th transmission round?
 - During what transmission round is the 70th segment sent?
 - Assuming a packet loss is detected after the 26th round by the receipt of a triple duplicate ACK, what will be the values of the congestion window size and of `ssthresh`?

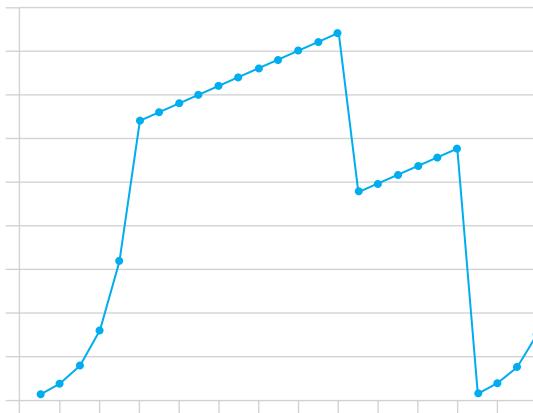


Figure 3.58 ♦ TCP window size as a function of time



VideoNote

Examining the
behavior of TCP

- j. Suppose TCP Tahoe is used (instead of TCP Reno), and assume that triple duplicate ACKs are received at the 16th round. What are the `ssthresh` and the congestion window size at the 19th round?
- k. Again suppose TCP Tahoe is used, and there is a timeout event at 22nd round. How many packets have been sent out from 17th round till 22nd round, inclusive?
- P41. Refer to Figure 3.56, which illustrates the convergence of TCP's AIMD algorithm. Suppose that instead of a multiplicative decrease, TCP decreased the window size by a constant amount. Would the resulting AIAD algorithm converge to an equal share algorithm? Justify your answer using a diagram similar to Figure 3.56.
- P42. In Section 3.5.4, we discussed the doubling of the timeout interval after a timeout event. This mechanism is a form of congestion control. Why does TCP need a window-based congestion-control mechanism (as studied in Section 3.7) in addition to this doubling-timeout-interval mechanism?
- P43. Host A is sending an enormous file to Host B over a TCP connection. Over this connection there is never any packet loss and the timers never expire. Denote the transmission rate of the link connecting Host A to the Internet by R bps. Suppose that the process in Host A is capable of sending data into its TCP socket at a rate S bps, where $S = 10 \cdot R$. Further suppose that the TCP receive buffer is large enough to hold the entire file, and the send buffer can hold only one percent of the file. What would prevent the process in Host A from continuously passing data to its TCP socket at rate S bps? TCP flow control? TCP congestion control? Or something else?
Elaborate.
- P44. Consider sending a large file from a host to another over a TCP connection that has no loss.
- Suppose TCP uses AIMD for its congestion control without slow start. Assuming `cwnd` increases by 1 MSS every time a batch of ACKs is received and assuming approximately constant round-trip times, how long does it take for `cwnd` increase from 6 MSS to 12 MSS (assuming no loss events)?
 - What is the average throughout (in terms of MSS and RTT) for this connection up through time = 6 RTT?
- P45. Recall the macroscopic description of TCP throughput. In the period of time from when the connection's rate varies from $W/(2 \cdot \text{RTT})$ to W/RTT , only one packet is lost (at the very end of the period).
- Show that the loss rate (fraction of packets lost) is equal to

$$L = \text{loss rate} = \frac{1}{\frac{3}{8} W^2 + \frac{3}{4} W}$$

- b. Use the result above to show that if a connection has loss rate L , then its average rate is approximately given by

$$\approx \frac{1.22 \cdot MSS}{RTT \sqrt{L}}$$

- P46. Consider that only a single TCP (Reno) connection uses one 10Mbps link which does not buffer any data. Suppose that this link is the only congested link between the sending and receiving hosts. Assume that the TCP sender has a huge file to send to the receiver, and the receiver's receive buffer is much larger than the congestion window. We also make the following assumptions: each TCP segment size is 1,500 bytes; the two-way propagation delay of this connection is 150 msec; and this TCP connection is always in congestion avoidance phase, that is, ignore slow start.
- a. What is the maximum window size (in segments) that this TCP connection can achieve?
 - b. What is the average window size (in segments) and average throughput (in bps) of this TCP connection?
 - c. How long would it take for this TCP connection to reach its maximum window again after recovering from a packet loss?
- P47. Consider the scenario described in the previous problem. Suppose that the 10Mbps link can buffer a finite number of segments. Argue that in order for the link to always be busy sending data, we would like to choose a buffer size that is at least the product of the link speed C and the two-way propagation delay between the sender and the receiver.
- P48. Repeat Problem 43, but replacing the 10 Mbps link with a 10 Gbps link. Note that in your answer to part c, you will realize that it takes a very long time for the congestion window size to reach its maximum window size after recovering from a packet loss. Sketch a solution to solve this problem.
- P49. Let T (measured by RTT) denote the time interval that a TCP connection takes to increase its congestion window size from $W/2$ to W , where W is the maximum congestion window size. Argue that T is a function of TCP's average throughput.
- P50. Consider a simplified TCP's AIMD algorithm where the congestion window size is measured in number of segments, not in bytes. In additive increase, the congestion window size increases by one segment in each RTT. In multiplicative decrease, the congestion window size decreases by half (if the result is not an integer, round down to the nearest integer). Suppose that two TCP connections, C_1 and C_2 , share a single congested link of speed 30 segments per second. Assume that both C_1 and C_2 are in the congestion avoidance

phase. Connection C_1 's RTT is 50 msec and connection C_2 's RTT is 100 msec. Assume that when the data rate in the link exceeds the link's speed, all TCP connections experience data segment loss.

- a. If both C_1 and C_2 at time t_0 have a congestion window of 10 segments, what are their congestion window sizes after 1000 msec?
 - b. In the long run, will these two connections get the same share of the bandwidth of the congested link? Explain.
- P51. Consider the network described in the previous problem. Now suppose that the two TCP connections, C_1 and C_2 , have the same RTT of 100 msec. Suppose that at time t_0 , C_1 's congestion window size is 15 segments but C_2 's congestion window size is 10 segments.
- a. What are their congestion window sizes after 2200msec?
 - b. In the long run, will these two connections get about the same share of the bandwidth of the congested link?
 - c. We say that two connections are synchronized, if both connections reach their maximum window sizes at the same time and reach their minimum window sizes at the same time. In the long run, will these two connections get synchronized eventually? If so, what are their maximum window sizes?
 - d. Will this synchronization help to improve the utilization of the shared link? Why? Sketch some idea to break this synchronization.
- P52. Consider a modification to TCP's congestion control algorithm. Instead of additive increase, we can use multiplicative increase. A TCP sender increases its window size by a small positive constant a ($0 < a < 1$) whenever it receives a valid ACK. Find the functional relationship between loss rate L and maximum congestion window W . Argue that for this modified TCP, regardless of TCP's average throughput, a TCP connection always spends the same amount of time to increase its congestion window size from $W/2$ to W .
- P53. In our discussion of TCP futures in Section 3.7, we noted that to achieve a throughput of 10 Gbps, TCP could only tolerate a segment loss probability of $2 \cdot 10^{-10}$ (or equivalently, one loss event for every 5,000,000,000 segments). Show the derivation for the values of $2 \cdot 10^{-10}$ (1 out of 5,000,000) for the RTT and MSS values given in Section 3.7. If TCP needed to support a 100 Gbps connection, what would the tolerable loss be?
- P54. In our discussion of TCP congestion control in Section 3.7, we implicitly assumed that the TCP sender always had data to send. Consider now the case that the TCP sender sends a large amount of data and then goes idle (since it has no more data to send) at t_1 . TCP remains idle for a relatively long period of time and then wants to send more data at t_2 . What are the advantages and disadvantages of having TCP use the `cwnd` and `ssthresh` values from t_1 when starting to send data at t_2 ? What alternative would you recommend? Why?

- P55. In this problem we investigate whether either UDP or TCP provides a degree of end-point authentication.
- Consider a server that receives a request within a UDP packet and responds to that request within a UDP packet (for example, as done by a DNS server). If a client with IP address X spoofs its address with address Y, where will the server send its response?
 - Suppose a server receives a SYN with IP source address Y, and after responding with a SYNACK, receives an ACK with IP source address Y with the correct acknowledgment number. Assuming the server chooses a random initial sequence number and there is no “man-in-the-middle,” can the server be certain that the client is indeed at Y (and not at some other address X that is spoofing Y)?
- P56. In this problem, we consider the delay introduced by the TCP slow-start phase. Consider a client and a Web server directly connected by one link of rate R . Suppose the client wants to retrieve an object whose size is exactly equal to $15S$, where S is the maximum segment size (MSS). Denote the round-trip time between client and server as RTT (assumed to be constant). Ignoring protocol headers, determine the time to retrieve the object (including TCP connection establishment) when
- $4S/R > S/R + RTT > 2S/R$
 - $S/R + RTT > 4S/R$
 - $S/R > RTT$.



Programming Assignments

Implementing a Reliable Transport Protocol

In this laboratory programming assignment, you will be writing the sending and receiving transport-level code for implementing a simple reliable data transfer protocol. There are two versions of this lab, the alternating-bit-protocol version and the GBN version. This lab should be fun—your implementation will differ very little from what would be required in a real-world situation.

Since you probably don't have standalone machines (with an OS that you can modify), your code will have to execute in a simulated hardware/software environment. However, the programming interface provided to your routines—the code that would call your entities from above and from below—is very close to what is done in an actual UNIX environment. (Indeed, the software interfaces described in this programming assignment are much more realistic than the infinite loop senders and receivers that many texts describe.) Stopping and starting

timers are also simulated, and timer interrupts will cause your timer handling routine to be activated.

The full lab assignment, as well as code you will need to compile with your own code, are available at this book's Web site: <http://www.awl.com/kurose-ross>.



Wireshark Lab: Exploring TCP

In this lab, you'll use your Web browser to access a file from a Web server. As in earlier Wireshark labs, you'll use Wireshark to capture the packets arriving at your computer. Unlike earlier labs, you'll *also* be able to download a Wireshark-readable packet trace from the Web server from which you downloaded the file. In this server trace, you'll find the packets that were generated by your own access of the Web server. You'll analyze the client- and server-side traces to explore aspects of TCP. In particular, you'll evaluate the performance of the TCP connection between your computer and the Web server. You'll trace TCP's window behavior, and infer packet loss, retransmission, flow control and congestion control behavior, and estimated roundtrip time.

As is the case with all Wireshark labs, the full description of this lab is available at this book's Web site, <http://www.awl.com/kurose-ross>.



Wireshark Lab: Exploring UDP

In this short lab, you'll do a packet capture and analysis of your favorite application that uses UDP (for example, DNS or a multimedia application such as Skype). As we learned in Section 3.3, UDP is a simple, no-frills transport protocol. In this lab, you'll investigate the header fields in the UDP segment as well as the checksum calculation.

As is the case with all Wireshark labs, the full description of this lab is available at this book's Web site, <http://www.awl.com/kurose-ross>.

AN INTERVIEW WITH...

Van Jacobson

Van Jacobson is a Research Fellow at PARC. Prior to that, he was co-founder and Chief Scientist of Packet Design. Before that, he was Chief Scientist at Cisco. Before joining Cisco, he was head of the Network Research Group at Lawrence Berkeley National Laboratory and taught at UC Berkeley and Stanford. Van received the ACM SIGCOMM Award in 2001 for outstanding lifetime contribution to the field of communication networks and the IEEE Kobayashi Award in 2002 for "contributing to the understanding of network congestion and developing congestion control mechanisms that enabled the successful scaling of the Internet". He was elected to the U.S. National Academy of Engineering in 2004.



Please describe one or two of the most exciting projects you have worked on during your career. What were the biggest challenges?

School teaches us lots of ways to find answers. In every interesting problem I've worked on, the challenge has been finding the right question. When Mike Karels and I started looking at TCP congestion, we spent months staring at protocol and packet traces asking "Why is it failing?". One day in Mike's office, one of us said "The reason I can't figure out why it fails is because I don't understand how it ever worked to begin with." That turned out to be the right question and it forced us to figure out the "ack clocking" that makes TCP work. After that, the rest was easy.

More generally, where do you see the future of networking and the Internet?

For most people, the Web is the Internet. Networking geeks smile politely since we know the Web is an application running over the Internet but what if they're right? The Internet is about enabling conversations between pairs of hosts. The Web is about distributed information production and consumption. "Information propagation" is a very general view of communication of which "pairwise conversation" is a tiny subset. We need to move into the larger tent. Networking today deals with broadcast media (radios, PONs, etc.) by pretending it's a point-to-point wire. That's massively inefficient. Terabits-per-second of data are being exchanged all over the World via thumb drives or smart phones but we don't know how to treat that as "networking". ISPs are busily setting up caches and CDNs to scalably distribute video and audio. Caching is a necessary part of the solution but there's no part of today's networking—from Information, Queuing or Traffic Theory down to the Internet protocol specs—that tells us how to engineer and deploy it. I think and hope that over the next few years, networking will evolve to embrace the much larger vision of communication that underlies the Web.

What people inspired you professionally?

When I was in grad school, Richard Feynman visited and gave a colloquium. He talked about a piece of Quantum theory that I'd been struggling with all semester and his explanation was so simple and lucid that what had been incomprehensible gibberish to me became obvious and inevitable. That ability to see and convey the simplicity that underlies our complex world seems to me a rare and wonderful gift.

What are your recommendations for students who want careers in computer science and networking?

It's a wonderful field—computers and networking have probably had more impact on society than any invention since the book. Networking is fundamentally about connecting stuff, and studying it helps you make intellectual connections: Ant foraging & Bee dances demonstrate protocol design better than RFCs, traffic jams or people leaving a packed stadium are the essence of congestion, and students finding flights back to school in a post-Thanksgiving blizzard are the core of dynamic routing. If you're interested in lots of stuff and want to have an impact, it's hard to imagine a better field.

This page intentionally left blank

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 and application layers, 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 examine two broad approaches towards structuring network-layer packet delivery, the dataagram and the virtual-circuit model, 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 ~~forwarding and routing~~ functions of the network layer. Forwarding involves the transfer of a 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. This will be an important distinction to keep in mind as you progress through this chapter.

In order to deepen our understanding of packet forwarding, we’ll look briefly at a router’s 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 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, shortest paths) 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 may 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 through 4.2, covers network-layer functions and services. The second part, Sections 4.3 through 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. Let’s 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, encodes each segment into a datagram (that is, a network-layer packet), and then forwards 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 function 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, without the upper layers above the network layer, because (except for control purposes) these layers do not run application- and transport-layer protocols such as those we examined in Chapters 2 and 3.

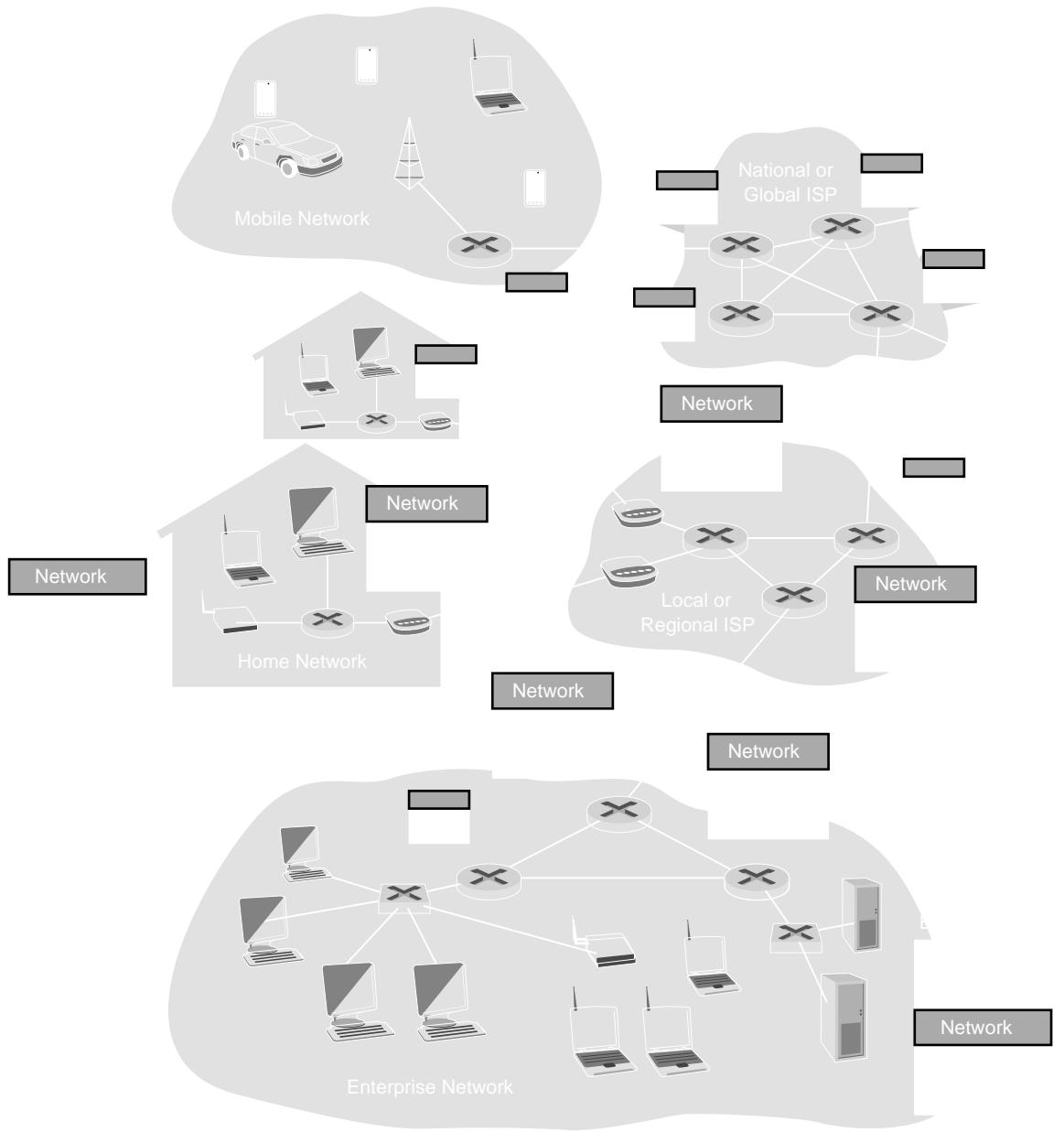


Figure 4.1 The network layer

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 transfer 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 Host H2. In Section 4.3, we’ll look inside a router and examine how a packet is actually forwarded from an input link to an output link within a router.
- € **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 routes 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 chapter. **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 taken by our traveler back in Section 1.3.1. During this trip, our driver passes 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, each path consisting of a series of road segments connected at interchanges.

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 value stored in the forwarding table entry for that header indicates the router’s outgoing link interface through which that packet is to be forwarded. Depending on the network-layer protocol, the header value 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 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 its forwarding function in much greater detail.

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

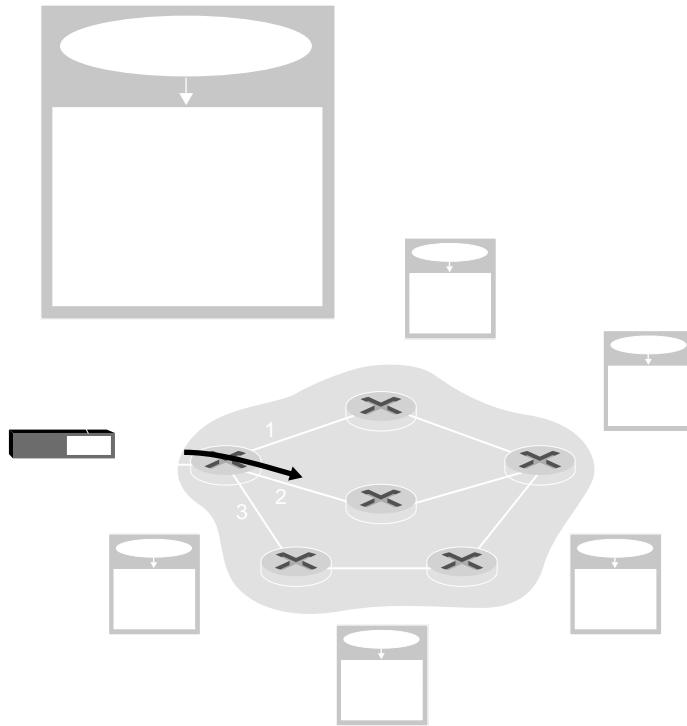


Figure 4.2 Routing algorithms determine values in forwarding tables

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 loading routing information to each of the routers) or decentralized (i.e., 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 unrealistically 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 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 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 call a **packet switch** to mean a general packet-switching device that transports a packet from input link interface to output link interface, according to the value in the header of the packet. Some packet switches, called **layer switches** (examined in Chapter 5), base their forwarding decision on values in the fields of the layer frame; switches are thus referred to as link-layer (layer 2) devices. Other switches, called **routers**, base their forwarding decision on the value in the network layer field. Routers are thus network-layer (layer 3) devices, but must also implement layer 2 protocols as well, since layer 3 devices require the services of layer 2 to implement their (layer 3) functionality. (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.) To confuse matters, marketing literature refers to "layer 3 switches" for routers with Ethernet interfaces, but these are still layer 3 devices. 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, network-layer architectures, for example, ATM, frame relay, and MPLS (which we will study in Section 5.8).....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 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 to the network layer at the sending host), can the transport layer rely 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

provide any feedback about congestion in the network? What is the abstraction (properties) of the channel connecting the transport layer in the sending and receiving hosts? The answers to these questions and others are determined by the service provided by the network layer. The network service model defines the characteristics of end-to-end transport of packets 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, 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 **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. 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 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 sender 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. 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

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

no service at all. With best-effort service, timing between packets is not guaranteed 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. By definition, a network that delivered all 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.

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 2012, 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 two alternatives do exist to the Internet's best-effort model. Two of the more important 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 connection whose properties are the same as if a dedicated fixed-bandwidth transmission link existed between sending and receiving hosts. With CBR service, a connection's bandwidth is carried across the network in such a way that a cell's end-to-end delay, the variability in a cell's end-to-end delay (that is, the jitter), and the number 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 switch when the CBR connection is first established.

- € Available bit rate (ABR) ATM network service With the Internet offering something 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 mentioned earlier, cells may be lost under ABR service. Unlike in the Internet, however, the 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 the form 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 maximum 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 between two processes. For example, the Internet's transport layer provides each application a choice between two services: a connectionless service; or TCP, a connection-oriented service. In a similar manner, the network layer can provide connectionless service or connection service between hosts. Network-layer connection and connectionless services in many ways parallel the 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 handshaking preliminaries.

Although the network-layer connection and connectionless services have 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 for the transport layer. In the transport layer these services are process-to-process services provided by the transport layer for 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

implemented at the edge of the network in the end systems; we'll see shortly that 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

While the Internet is a datagram network, 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 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 replaces 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 22, 23, and 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 22; 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



Figure 4.3 A simple virtual circuit network

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, data 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, during VC setup, the 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 establish the connection state in router tables) are known as **signaling messages**, and the protocol

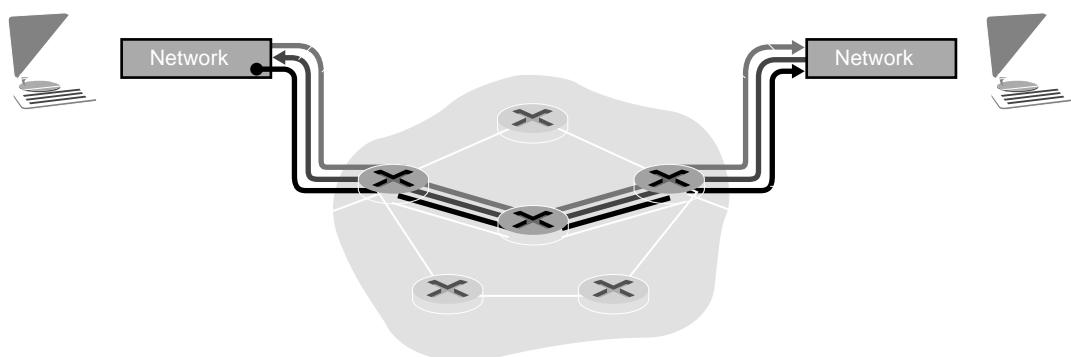


Figure 4.4 Virtual-circuit setup

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 1995] 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 encodes the packet with the address of the destination end system and then pops the packet into the network. As shown in Figure 4.5, there is no VC setup and routers do not maintain any VC state information (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 its forwarding table. The router then intentionally forwards the packet to that 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 out of the question.

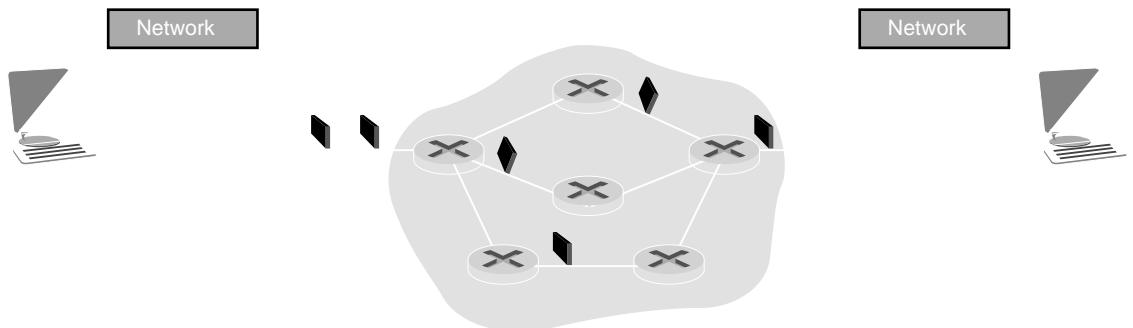


Figure 4.5 Datagram network

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 000101 11111111	0
11001000 00010111 00011000 00000000 through 11001000 00010111 0001100 01111111	1
11001000 00010111 00011001 00000000 through 11001000 00010111 0 01111111	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 **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 destination address is 11001000 00010111 00010110 10100001; because the 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 in the table, 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 **longest prefix matching rule**; that is, it finds the longest matching entry in the table and forwards the packet to

interface associated with the longest prefix match. We'll see exactly why the longest prefix-matching rule is used when we study Internet addressing in detail 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 much faster 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 et al. 2002] 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 virtual circuit as a central organizing principle has its roots in the telephony network, which uses real circuits. With call setup and per-call state being maintained by switches and 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 consistent with keeping with its telephony heritage. Telephone networks, by necessity, had to keep complexity within the network, since they were connecting dumb end-system devices such as rotary telephones. (For those too young to know, a rotary telephone 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, 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 resolution) is then implemented at a higher layer, in the end systems. This is very different from the model of the telephone network, with some interesting consequences:

- € Since the resulting Internet network-layer service model makes minimal service guarantees, it imposes minimal requirements on the network layer. This makes it easier to interconnect networks that use very different link-layer technologies (for example, satellite, Ethernet, fiber, or radio) that 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 network infrastructure services such as the DNS are implemented in servers at the network edge. The ability to add a new service simply by adding a host to the network and defining a new application-layer protocol (such as HTTP) has allowed new Internet applications such as the Web to be deployed in a remarkably short period of time.

4.3 What’s Inside a Router?

Now that we’ve overviewed the network layer’s services and functions, let our attention to its forwarding function, “the actual transfer of packets from router’s incoming links to the appropriate outgoing links at that route.” We already took a brief look at a few aspects of forwarding in Section 4.2, namely addressing and longest prefix matching. We mention here in passing that the terms “forwarding” and “switching” are often used interchangeably by computer-network researchers and practitioners; we’ll use both terms interchangeably in this textbook as well.

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

- € **Input ports.** An input port performs several key functions. It performs the physical layer function of terminating an incoming physical link at a router. This is shown in the leftmost box of the input port and the rightmost box of the output port in Figure 4.6. An input port also performs link-layer functions needed to interoperate with the link layer at the other side of the incoming link; this is represented by the middle boxes in the input and output port. Perhaps most crucially, the lookup function is also performed at the input port. This will occur in the rightmost box of the input port. It is here that the forwarding table is consulted to determine the router output port to which the arriving packet will be forwarded via the switching fabric. Control packets (for example, packets carrying routing protocol information) are forwarded from an input port to the routing processor. Note that the term “port,” referring to the physical input and output router interfaces, is distinct from the software ports associated with network application sockets discussed in Chapters 2 and 3.
- € **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 packets received from the switching fabric and transmits these packets on the outgoing link by performing the necessary link-layer and physical-layer functions. When a link is bidirectional (that is,

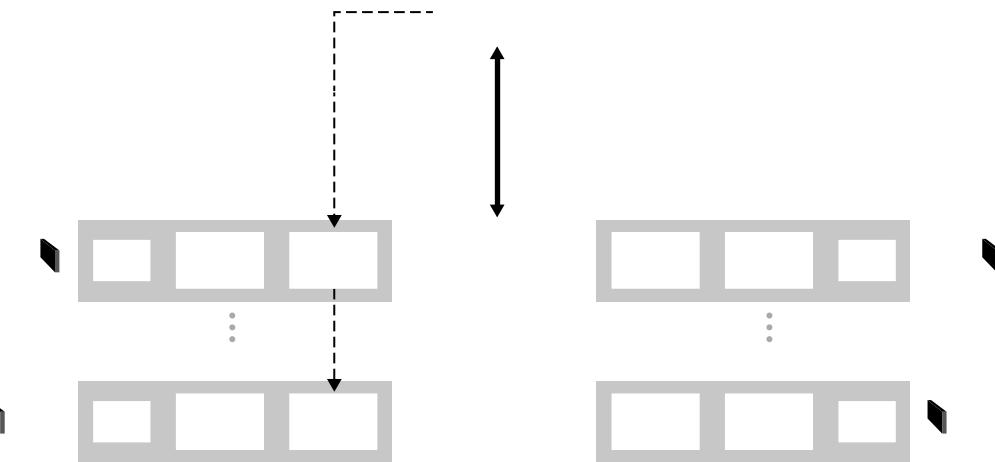


Figure 4.6 Router architecture

carries traffic in both directions), an output port will typically be paired with an input port for that link on the same line card (a printed circuit board containing one or more input ports, which is connected to the switching fabric).

- € Routing processor The routing processor executes the routing protocols (which we'll study in Section 4.6), maintains routing tables and attached link state information, and computes the forwarding table for the router. It also performs network management functions that we'll study in Chapter 9.

Recall that in Section 4.1.1 we distinguished between a router's forwarding and control functions. A router's input ports, output ports, and switching fabric together implement the forwarding function and are almost always implemented in hardware, as shown in Figure 4.6. These forwarding functions are sometimes collectively referred to as the router forwarding plane. To appreciate why a hardware implementation is needed, consider that with a 10 Gbps input link and a 64-byte IP datagram, the input port has only 51.2 ns to process the datagram before another datagram may arrive. If ports are combined on a line card (as is often done in practice), the datagram-processing pipeline must operate much faster, far too fast for software implementation. Forwarding plane hardware can be implemented either using a router vendor's own hardware designs, or by using purchased merchant-silicon chips (e.g., as sold by companies such as Intel and Broadcom).

While the forwarding plane operates at the nanosecond time scale, a router's control functions, executing the routing protocols, responding to attached link

go up or down, and performing management functions such as those we'll Chapter 9, operate at the millisecond or second timescale. The control plane functions are usually implemented in software and execute on the processor (typically a traditional CPU).

Before delving into the details of a router's control and data plane, let's re our analogy of Section 4.1.1, where packet forwarding was compared to cars and leaving an interchange. Let's suppose that the interchange is a roundabout before a car enters the roundabout, a bit of processing is required,,the car stops at an entry station and indicates its final destination (not at the local roundabout, but mate destination of its journey). An attendant at the entry station looks up the fi tination, determines the roundabout exit that leads to that final destination, and driver which roundabout exit to take. The car enters the roundabout (which filled with other cars entering from other input roads and heading to other rour exits) and eventually leaves at the prescribed roundabout exit ramp, where encounter other cars leaving the roundabout at that exit.

We can recognize the principal router components in Figure 4.6 in this analogy,,the entry road and entry station correspond to the input port (with a lo function to determine to local outgoing port); the roundabout corresponds switch fabric; and the roundabout exit road corresponds to the output port. In this analogy, it's instructive to consider where bottlenecks might occur. What happens if cars arrive blazingly fast (for example, the roundabout is in Germany or Italy!) but the station attendant is slow? How fast must the attendant work to there's no backup on an entry road? Even with a blazingly fast attendant, what happens if cars traverse the roundabout slowly,,can backups still occur? And what happens if most of the entering cars all want to leave the roundabout at the same exit ramp,,can backups occur at the exit ramp or elsewhere? How should the roundabout operate if we want to assign priorities to different cars, or block some cars from entering the roundabout in the first place? These are all analogouical questions faced by router and switch designers.

In the following subsections, we'll look at router functions in more detail. Belding et al. 2008; Chao 2001; Chuang 2005; Turner 1988; McKeown 1997a; Partridge 1997 provide a discussion of specific router architectures. For concreteness, the discussion assumes a datagram network in which 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 quite similar for a virtual circuit network.

4.3.1 Input Processing

A more detailed view of input processing is given in Figure 4.7. As discussed earlier, the input port's line termination function and link-layer processing implement the physical and link layers for that individual input link. The lookup performed in the input port is central to the router's operation,,it is here that the router uses the forwarding table to look up the output port to which an arriving packet will be forwarded.



CASE HISTORY

CISCO SYSTEMS: DOMINATING THE NETWORK CORE

As of this writing 2012, Cisco employs more than 65,000 people. 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, the primary adopters of the Internet at that time. 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 caching, Ethernet switch, datacenter infrastructure, video conferencing, and voice-over IP products and services. However, Cisco is facing increased international competition, including from Huawei, a rapidly growing Chinese network-gear company. Other sources of competition for Cisco in the router and switched Ethernet space include Alcatel-Lucent and Juniper.

forwarded via the switching fabric. The forwarding table is computed and updated by the routing processor, with a shadow copy typically stored at each input port. The forwarding table is copied from the routing processor to the line cards over a separate bus (e.g., a PCI bus) indicated by the dashed line from the routing processor to the input line cards in Figure 4.6. With a shadow copy, forwarding decisions can be made locally, at each input port, without invoking the centralized routing processor on a per-packet basis and thus avoiding a centralized processing bottleneck.

Given the existence of a forwarding table, lookup is conceptually simple, we just search through the forwarding table looking for the longest prefix match, as described



Figure 4.7 Input port processing

in Section 4.2.2. But at Gigabit transmission rates, this lookup must be performed in nanoseconds (recall our earlier example of a 10 Gbps link and a 64-byte IP datagram). Thus, not only must lookup be performed in hardware, but techniques beyond a simple linear search through a large table are needed; surveys of fast lookup algorithms can be found in [Gupta 2001, Ruiz-Sanchez 2001]. Special attention must also be paid to memory access times, resulting in designs with embedded on-chip DRAM and faster SRAM (used as a DRAM cache) memories. Ternary Content Address Memories (TCAM) are also often used for lookup. With a TCAM, a 32-bit IP address is presented to the memory, which returns the content of the forwarding table entry for that address in initially constant time. The Cisco 8500 has a 64K CAM for each input port.

Once a packet's output port has been determined via the lookup, the packet can be sent into the switching fabric. In some designs, a packet may be temporarily blocked from entering the switching fabric if packets from other input ports are currently using the fabric. A blocked packet will be queued at the input port and scheduled to cross the fabric at a later point in time. We'll take a closer look at blocking, queuing, and scheduling of packets (at both input ports and output ports) in Section 4.3.4. Although •lookup• is arguably the most important action in input port processing, many other actions must be taken: (1) physical- and link-layer processing must occur, as discussed above; (2) the packet's version number, destination IP address, and time-to-live field, all of which we'll study in Section 4.4.1, must be checked and the latter two fields rewritten; and (3) counters used for network management (such as the number of IP datagrams received) must be updated.

Let's close our discussion of input port processing by noting that the input steps of looking up an IP address (•match•) then sending the packet into the switching fabric (•action•) is a specific case of a more general •match plus action• abstraction that is performed in many networked devices, not just routers. In link-layer switches (covered in Chapter 5), link-layer destination addresses are looked up and actions may be taken in addition to sending the frame into the switching fabric to the output port. In firewalls (covered in Chapter 8), devices that filter out selected incoming packets, an incoming packet whose header matches a given criteria (a combination of source/destination IP addresses and transport-layer port numbers) can be prevented from being forwarded (action). In a network address translator (NAT, covered in Section 4.4), an incoming packet whose transport-layer port number matches a given value will have its port number rewritten before forwarding (action). Thus, the •match plus action• abstraction is both powerful and prevalent in network devices.

4.3.2 Switching

The switching fabric is at the very heart of a router, as it is through this fabric that the packets are actually switched (that is, forwarded) from an input port to an output port. Switching can be accomplished in a number of ways, as shown in Figure 4.10.

- € **Switching via memory.** The simplest, earliest routers were traditional computers with switching between input and output ports being done under direct control of the CPU.

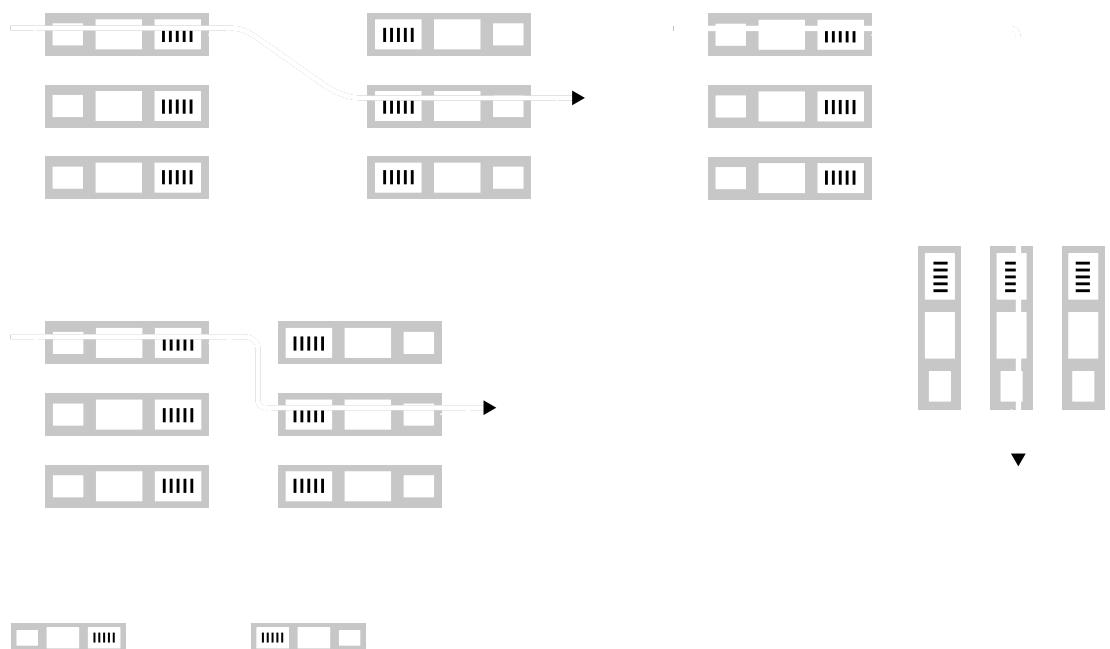


Figure 4.8 Three switching techniques

the CPU (routing processor). Input and output ports functioned as traditional 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. In this scenario, 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 rate at which packets are transferred from input ports to output ports) is no less than $B/2$. Note also that two packets cannot be forwarded at the same time, even if they have different destination ports, since only one memory read operation over the shared system bus can be done at a time.

Many modern routers switch via memory. A major difference from early routers, however, is that the lookup of the destination address and the storing of the packet into the appropriate memory location are performed by processing on the line cards. In some ways, routers that switch via memory look very much like shared-memory multiprocessors, with the processing on a line card writing (writing) packets into the memory of the appropriate output port. Cisco's C 8500 series switches [Cisco 8500 2012] forward packets via a shared memory.

- € Switching via a bus In this approach, an input port transfers a packet directly to an output port over a shared bus, without intervention by the routing processor. This is typically done by having the input port pre-pend a switch-internal label (header) to the packet indicating the local output port to which this packet is being transferred, and transmitting the packet onto the bus. The packet is received by all output ports, but only the port that matches the label will keep the packet. The label is removed at the output port, as this label is only used within the switch to control the bus. If multiple packets arrive to the router at the same time, each at a different output port, all but one must wait since only one packet can cross the bus at a time. Since every packet must cross the single bus, the switching speed of the router is limited to the bus speed; in our roundabout analogy, this is as if the roundabout could only contain one car at a time. Nonetheless, switching via a bus is often sufficient for routers that operate in small local area and enterprise networks. The Cisco 12000 [Cisco Switches 2012] 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 processor multiprocessor computer architecture. A crossbar switch is an interconnection network consisting of N buses that connect M input ports to N output ports, as shown in Figure 4.8. Each vertical bus intersects each horizontal bus at a crosspoint that can be opened or closed at any time by the switch fabric controller (whose logic is part of the switching fabric itself). When a packet arrives from port A and needs to be forwarded to port Y, the switch controller closes the crosspoint at the intersection of busses A and Y, and port A then sends the packet onto its bus, which is picked up (only) by bus Y. Note that a packet from port B can be forwarded to port X at the same time, since the A-to-Y and B-to-X packets use different input and output busses. Thus, unlike the previous two switching approaches, crossbar networks are capable of forwarding multiple packets in parallel. However, if two packets enter the switch from two different input ports destined to the same output port, then one will have to wait at the input, since only one packet can be sent over any given bus at a time. More sophisticated interconnection networks use multiple stages of switch elements to allow packets from different input ports to proceed towards their output port at the same time through the switching fabric. See [Tobagi 1994] for a survey of switch architectures. Cisco 12000 family switches [Cisco 2012] use an interconnection network.

4.3.3 Output Processing

Output port processing, shown in Figure 4.9, takes packets that have been buffered at the output port's memory and transmits them over the output link. This involves selecting and de-queueing packets for transmission, and performing the necessary data-link layer and physical-layer transmission functions.

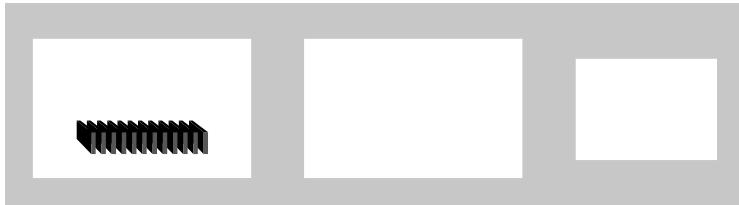


Figure 4.9 Output port processing

4.3.4 Where Does Queueing Occur?

If we consider input and output port functionality and the configurations shown in Figure 4.8, it's clear that packet queues may form at both the input and output ports, just as we identified cases where cars may wait at the inputs and outputs of the traffic intersection in our roundabout analogy. The location and extent of queuing (either at the input port queues or the output port queues) will depend on the load, the relative speed of the switching fabric, and the line speed. Let's now consider these queues in a bit more detail, since as these queues grow large, the router memory can eventually be exhausted and packet loss will occur when no memory is available to store arriving packets. Recall that in our earlier discussions, we saw that packets were lost within the network or dropped at a router. It is here, at the output queues within a router, where such packets are actually dropped and lost.

Suppose that the input and output line speeds (transmission rates) are all identical transmission rate R_{line} packets per second, and that there are N input ports and N output ports. To further simplify the discussion, let's assume that the packets have the same fixed length, and the packets arrive to input ports in a synchronous manner. That is, the time to send a packet on any link is equal to the time to receive a packet on any link, and during such an interval of time, either one packet can arrive on an input link. Define the switching fabric transfer rate R_{switch} as the rate at which packets can be moved from input port to output port. If R_{switch} is N times faster than R_{line} , then only negligible queuing will occur at the input ports. This is because even in the worst case, when all input lines are receiving packets, and all packets are to be forwarded to the same output port, a batch of N packets (one packet per input port) can be cleared through the switching fabric before the next batch arrives.

But what can happen at the output ports? Let's suppose R_{switch} is still N times faster than R_{line} . Once again, packets arriving at each of the N input ports are destined to the same output port. In this case, in the time it takes to send one packet onto the outgoing link, N new 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 arriving packets will have to queue (wait) for transmission on the outgoing link. Then N more packets can possibly arrive in the time it takes

transmit just one of the packets that had just previously been queued. And eventually, the number of queued packets can grow large enough to exhaustable memory at the output port, in which case packets are dropped.

Output port queuing is illustrated in Figure 4.10. At time t_0 , three packets have 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), the original packets have been transferred to the outgoing port and are queued for 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 incoming side of the switch; one of these packets is destined for this uppermost outgoing port.

Given that router buffers are needed to absorb the fluctuations in traffic load, a 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 $RTT \cdot C = 2.5$ Gbits of buffers. Recent

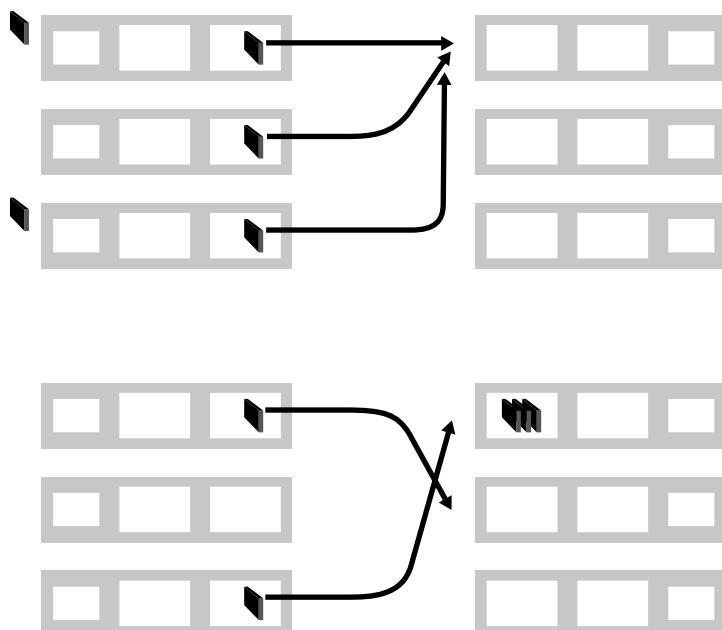


Figure 4.10 Output port queuing

theoretical and experimental efforts [Appenzeller 2004], however, suggest that there are a large number of TCP flows passing through a link, the amount of buffering needed is $\approx \text{RTT} \cdot C/N$. 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; Chik 2005; Beheshti 2008] provide very readable discussions of the buffer sizing problem from a theoretical, implementation, and operational standpoint.

A consequence of output port queuing is that at the output port scheduler 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, 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 will thus cover packet scheduling exclusively in Chapter 7. A discussion of output port packet scheduling disciplines [Cisco Queue 2012].

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 to remove one or more already-queued packets to make room for the newly arriving 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 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. [Chris 2001] and [Floyd 2012] provide overviews and pointers to additional reading.

If the switch fabric is not fast enough (relative to the input line speeds) to trans-

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 head of their input queues are destined for the same output queue, then one of the packets must be blocked and must wait at the input queue, while the switching fabric can transfer the other packet to a given output port at a time.

Figure 4.11 shows an example in which two packets (darkly shaded) at the heads of their input queues are destined for the same upper-right output port. Suppose 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. 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 the latter is the destination for the middle-right output port (the destination for the darkly shaded packet). This phenomenon is known as head-of-the-line (HOL) blocking in an input queued switch.

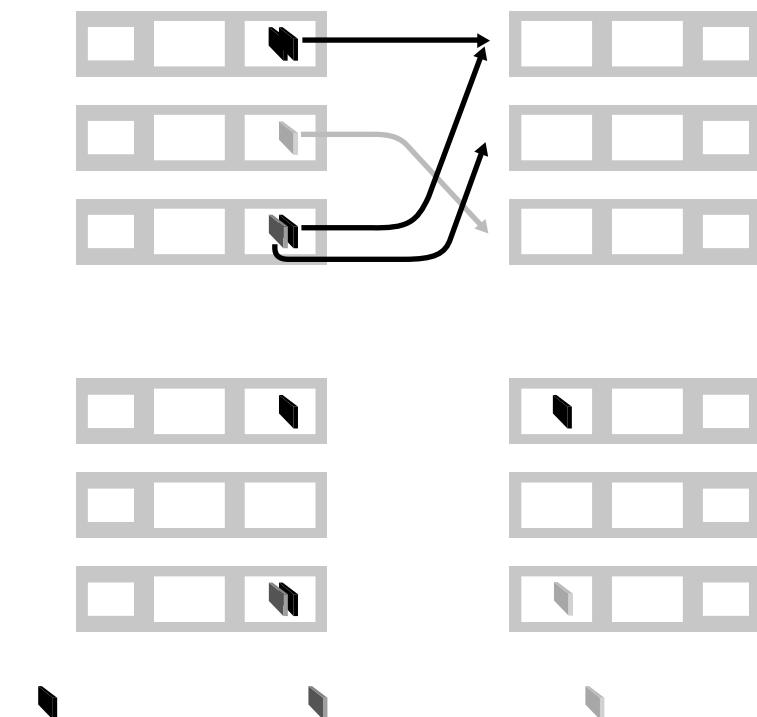


Figure 4.11 HOL blocking at an input queued switch

input-queued switch, a queued packet in an input queue must wait for traffic 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 the packet arrival rate on the input links reaches only 58 percent of their capacity. Number of solutions to HOL blocking are discussed in [McKeown 1997b].

4.3.5 The Routing Control Plane

In our discussion thus far and in Figure 4.6, we've implicitly assumed that the routing control plane fully resides and executes in a routing processor within the router. The network-wide routing control plane is thus decentralized, with different pieces (e.g., of a routing algorithm) executing at different routers and interacting by sending control messages to each other. Indeed, today's Internet routers and the routing algorithms we'll study in Section 4.6 operate in exactly this manner. Additionally, router and switch vendors bundle their hardware data plane and software control plane together into closed (but inter-operable) platforms in a vertically integrated product.

Recently, a number of researchers [Caesar 2005a, Casado 2009, McKeown 2008] have begun exploring new router control plane architectures in which the control plane is implemented in the routers (e.g., local measurement/reporting of link state, forwarding table installation and maintenance) along with the data plane, and part of the control plane can be implemented externally to the router (e.g., in a centralized server, which could perform route calculation). A well-defined API states how these two parts interact and communicate with each other. These researchers argue that separating the software control plane from the hardware data plane (with a minimal router-resident control plane) can simplify routing by reducing distributed routing calculation with centralized routing calculation, and enable network innovation by allowing different customized control planes to operate on fast hardware data planes.

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 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

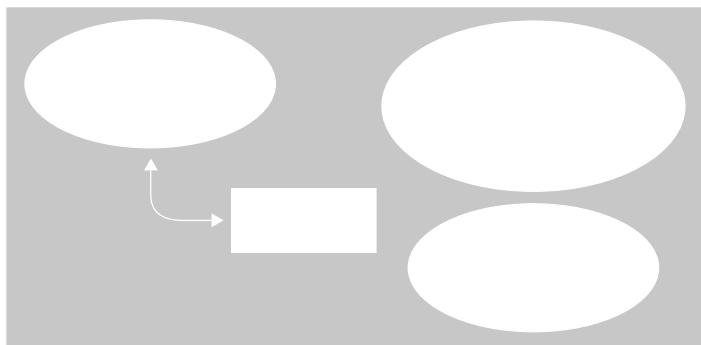


Figure 4.12 A look inside the Internet’s network layer

[RFC 791]. We’ll examine IP version 6 [RFC 2460; RFC 4291], which has 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 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 network-layer information. We’ll cover the Internet’s network-layer error 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 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.

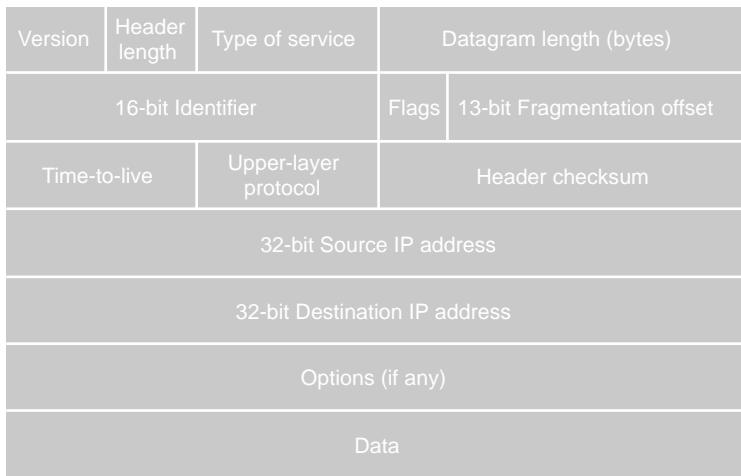


Figure 4.13 IPv4 datagram format

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) will be 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 used to determine where in the IP datagram the data actually begins. Most IPv4 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 prioritized for requiring low delay, high throughput, or reliability) to be distinguished from one another. 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 different levels of service in Chapter 7.

- € **Datagram length** This is the total length of the IP datagram (header plus payload) 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 IPv4 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 a 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.
- € **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 [IANA Protocol Numbers 2012]. Note that the protocol number in the IP header 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 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 received IP datagrams. The header checksum is computed by treating each byte in the header as a number and summing these numbers using 1's complement arithmetic. As discussed in Section 3.3, the 1's 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 if the checksum carried in the datagram header does not equal the computed sum. Routers typically discard datagrams for which an error has been detected. Note that the checksum must be recomputed and stored again at each router. The TTL field, and possibly the options field as well, may change. An interesting discussion of fast algorithms for computing the Internet checksum is given in [Kohno 1971]. A question often asked at this point is, why does TCP/IP perform header checksumming 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 Internet 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 destination host into the destination IP address field.

- ultimate destination into the destination IP address field. Often the source determines the destination address via a DNS lookup, as discussed in Chapter 4. We'll discuss IP addressing in detail in Section 4.4.2.
- € Options. The options fields allow an IP header to be extended. Header compression was meant to be used rarely, hence the decision to save overhead by 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 starts. Also, since some datagrams may require options processing and others not, the amount of time needed to process an IP datagram at a router can vary 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 payload field of the IP datagram. Finally, we come to the last and most important field, the payload field of the IP datagram in the first place! In most circumstances, the payload field of the IP datagram contains the transport-layer segment (TCP or UDP), which will 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 has a total of 40 bytes of header (20 bytes of IP header plus 20 bytes of TCP header) plus 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 others can carry only little packets. For example, Ethernet frames can carry up to 1500 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. It is 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 uses different link-layer protocols, and each of these protocols can have different MTUs.

To understand the forwarding issue better, imagine 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

The solution is to fragment the data in the IP datagram into two or more smaller datagrams, encapsulate each of these smaller IP datagrams in a separate frame; and send these frames over the outgoing link. Each of these small datagrams is referred to as a fragment.

Fragments need to be reassembled before they reach the transport layer application. Indeed, both TCP and UDP are expecting to receive complete, unfragmented segments from the network layer. The designers of IPv4 felt that reassembling fragments in the routers would introduce significant complication into the protocol and put a damper on router performance. (If you were a router, would you want to spend time 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 task 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 larger datagram. If some datagrams are fragments, it must further determine which datagram they came from, whether they have 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) added identification, flags, and fragmentation offset fields in the IP datagram header. When a datagram is created, the sending host stamps the datagram with an identification number, 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 same address, destination address, and identification number of the original datagram. When the destination receives a series of datagrams from the same sender, it can examine the identification numbers of the datagrams to determine which datagrams are actually fragments of the same larger datagram. Because of the 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 indicate where the fragment fits within the original IP datagram.

Figure 4.14 illustrates an example. A datagram of 4,000 bytes (20 byte header plus 3,980 bytes of IP payload) arrives at a router and must be forwarded on a link with an MTU of 1,500 bytes. This implies that the 3,980 data bytes of 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.

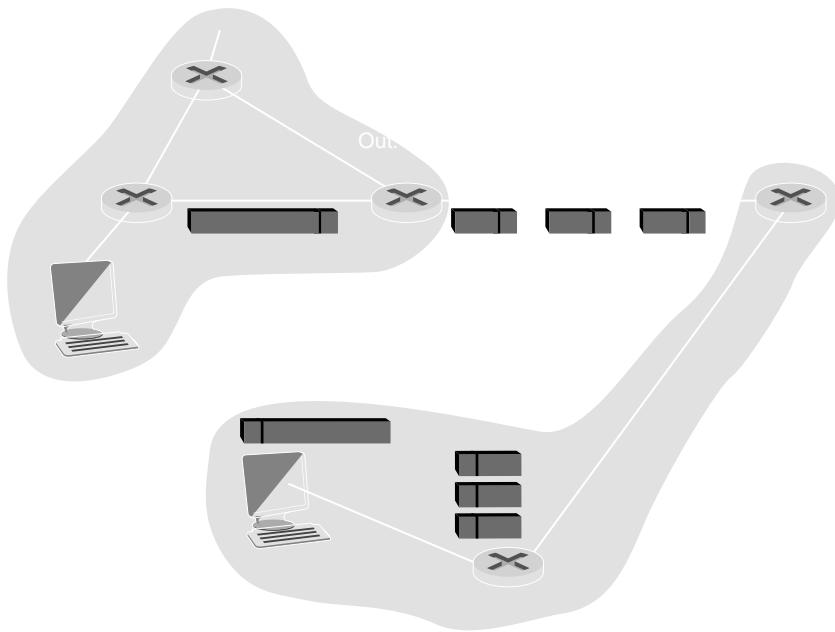


Figure 4.14 IP fragmentation and reassembly

Table 4.2 IP fragments

chapter, if TCP is being used at the transport layer, then TCP will recover from loss by having the source retransmit the data in the original datagram.

We have just learned that IP fragmentation plays an important role in bringing together the many disparate link-layer technologies. But fragmentation also has 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 small 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 value of zero. The target can collapse as it attempts to rebuild datagrams out of the separate 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 fragmented datagrams given the incoming datagram size, the MTU, and the incoming datagram in its entirety. The applet automatically generates the fragments for you. See 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 2012] and the first chapter of [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 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, the 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, which 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 2^{32} possible IP addresses. By approximating 2^{32} by 10^8 , it is easy to see that there

are about 4 billion possible IP addresses. These addresses are typically written in 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 next 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 fashion, 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 diagram, 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, as there are several details 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.x. That is, they all have the same leftmost 24 bits in their IP address. The four interfaces of the router are also interconnected to each other by a network backbone. This network contains no routers.

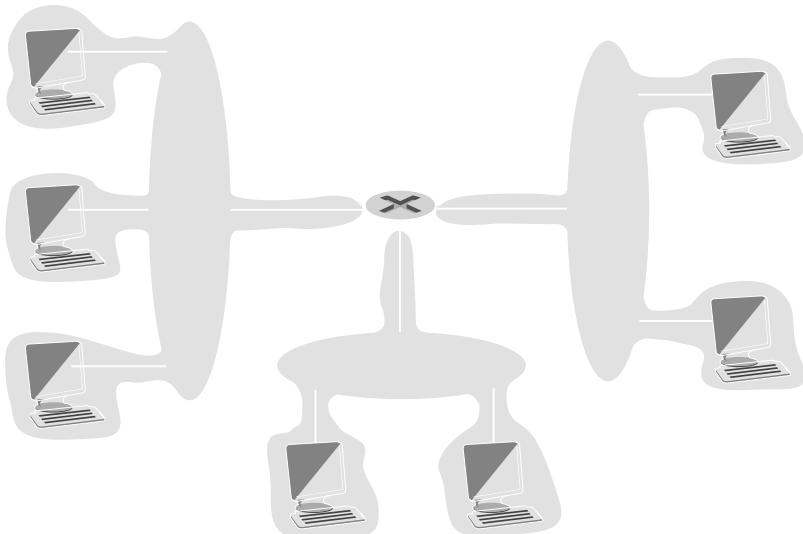


Figure 4.15 Interface addresses and subnets

could be interconnected by an Ethernet LAN, in which case the interfaces would be interconnected by an Ethernet switch (as we'll discuss in Chapter 5), or by a wireless access point (as we'll discuss in Chapter 6). We'll represent this routerless network by connecting these hosts as a cloud for now, and dive into the internals of such networks in Chapters 5 and 6.

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 a 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). Additional hosts attached to the 223.1.1.0/24 subnet would be expected 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 4.16, 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, similar to the subnets we encountered in Figure 4.15. But note that there are differences between this figure and Figure 4.15.

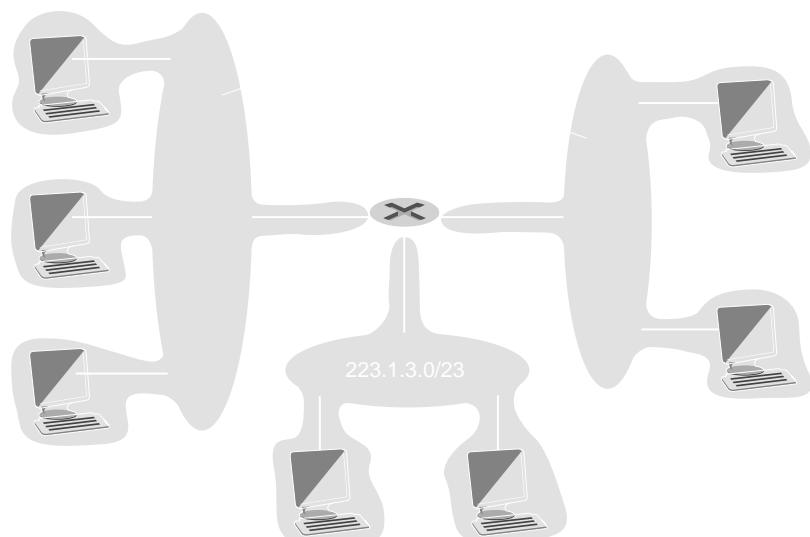


Figure 4.16 Subnet addresses

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 connect routers R3 and R1. For a general interconnected system of routers and switches 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 find three 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.

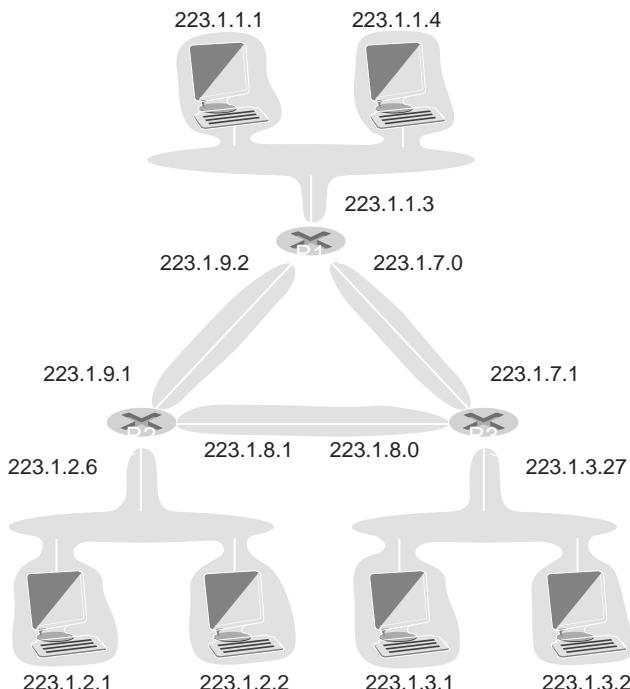


Figure 4.17 Three routers interconnecting six subnets

The Internet's address assignment strategy is known as Classless Interdomain Routing (CIDR „pronounced cider“) [RFC 4632]. CIDR generalizes the notion of 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 199.31.0.0/16, 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 199.31.0.0/16. 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.

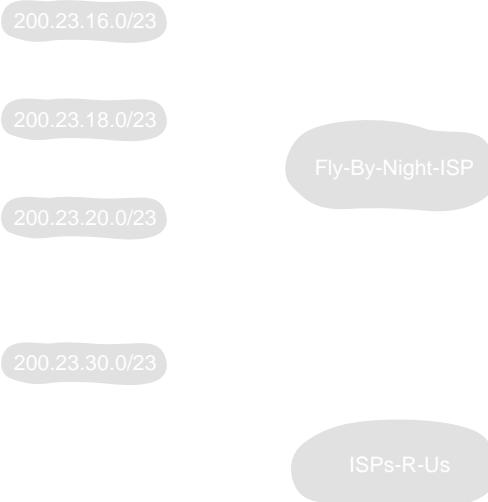


Figure 4.18 Hierarchical addressing and route aggregation

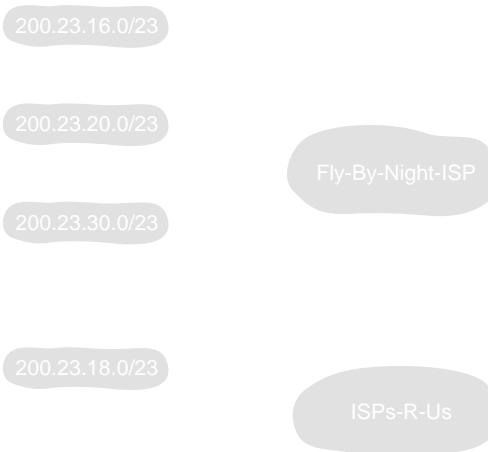


Figure 4.19 ISPs-R-Us has a more specific route to Organization 1

routing protocol in Section 4.6, we'll see that only the leading prefix bits are considered by routers outside the organization's network. That is, when a host outside the organization forwards a datagram whose destination address is within the organization, only the leading 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-k 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 the edge of the organization. These lower-order bits may (or may not) have an additional subnetting structure, such as that discussed above. For example, suppose the last 11 bits of the CIDRized address a.b.c.d/21 specify the organization's network 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 subnets within the organization, as discussed above. For example, a.b.c.d/24 might represent 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 problematical in supporting the rapidly growing number of organizations with small and medium-sized subnets. A class C (/24) subnet could accommodate only up to $2^{24} - 2 = 254$ hosts (two of the $2^8 = 256$ addresses are reserved for special use), too small for most organizations. However, a class B (/16) subnet, which supports up to 65,534 hosts, was too large. Under classful addressing, an organization with, say, 2,000 hosts 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 able to use only enough of the address space for up to 65,534 interfaces, leaving more than 65,000 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 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 in 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 host (such as a computer) is assigned an address from within the organization's block of addresses.

Obtaining a Block of Addresses

In order to obtain a block of IP addresses for use within an organization, 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. 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	<u>1001000 00010111 00010000 00000000</u>
Organization 0	200.23.16.0/23	<u>1001000 00010111 00010000 00000000</u>
Organization 1	200.23.18.0/23	<u>1001000 00010111 00010000 00000000</u>
Organization 2	200.23.20.0/23	<u>1001000 00010111 00010010 00000000</u>
...
Organization 7	200.23.30.0/23	<u>1001000 00010111 00011110 00000000</u>

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 Internet 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 organizations? Indeed there is! IP addresses are managed under the authority of the Internet Corporation for Assigned Names and Numbers (ICANN) [ICANN 2012], 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 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 2012]), which 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 addresses to the host and router interfaces in its organization. A system administrator will typically manually configure the IP addresses into the router (or, 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

given host receives the same IP address each time it connects to the network. A host may be assigned temporary IP addresses that will be different each time the host connects to the network. In addition to host IP address assignment, DHCP 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 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 just 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 occasion, the student will be connecting into a new subnet and hence will need a new address at each location. DHCP is ideally suited to this situation, as there are 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 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 400 addresses (for example, a block of the form a.b.c.d/23). As the hosts join and leave the network, 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 4.1) has 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 responsible for sending requests to the server. 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, *your Internet address* indicates the address being allocated to the 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.

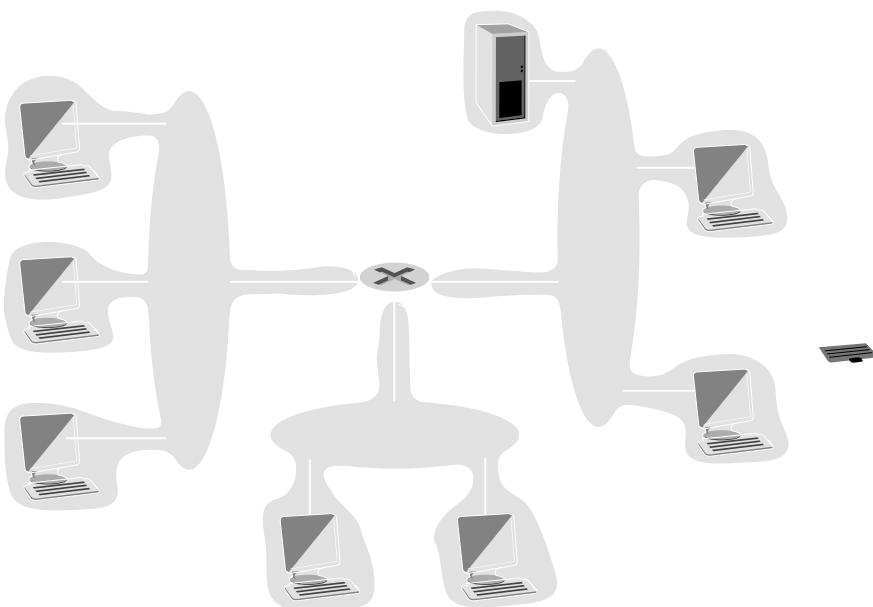


Figure 4.20 DHCP client-server scenario

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 a broadcast destination IP address of 255.255.255.255 and a “this host” source 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 (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) If several DHCP servers can be present on the subnet, the client may find the enviable position of being able to choose from among several offers. The server offer message contains the transaction ID of the received discover message, the proposed IP address for the client, the network mask, and the 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 2000].

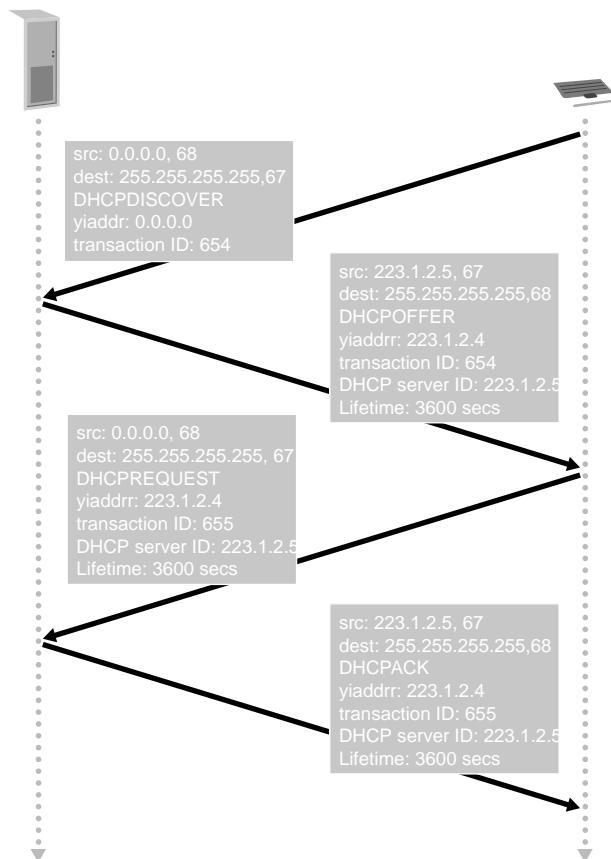


Figure 4.21 DHCP client-server interaction

- € **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

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 wireless network, 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, as most students (except those taking a computer networking class!) would have the desire to configure their laptops manually. From a mobility aspect, however, DHCP does have shortcomings. Since a new IP address is obtained from DHCP every time a node connects to a new subnet, a TCP connection to a remote application must 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 node to use a single permanent address as it moves between subnets. A detailed discussion about DHCP can be found in [Droms 2002] and [dhc 2012]. An open source reference implementation of DHCP is available from the Internet Systems Consortium [ISC 2012].

Network Address Translation (NAT)

Given our discussion about Internet addresses and the IPv4 datagram format, we are 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 impose a problem: whenever a SOHO wants to install a LAN to connect multiple machines, a range of IP 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 smartphones 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 space? 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 addressing that has found increasingly widespread use in such scenarios as network address translation(NAT) [RFC 2663; RFC 3022; Zhang 2007].

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, 10.0.0/24. The address space 10.0.0.0/8 is one of three portions of the IP space that is reserved in [RFC 1918] for a private network realm with private IP addresses, such as the home network in Figure 4.22. A network realm 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

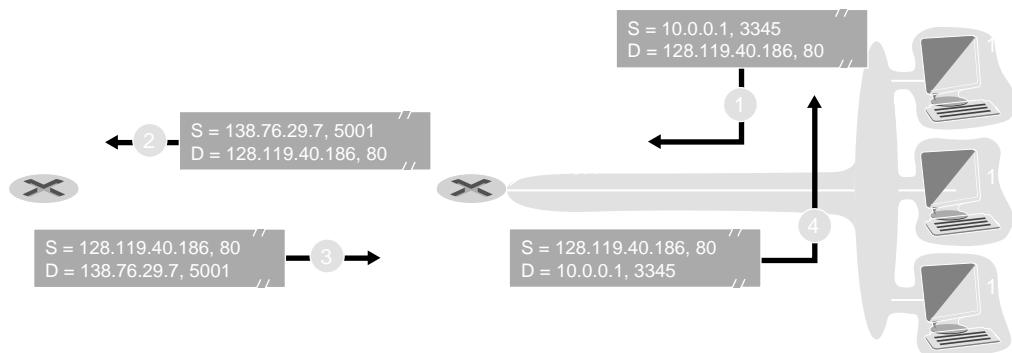


Figure 4.22 Network address translation

thousands of home networks, many using the same address space, 10. Devices within a given home network can send packets to each other using 10.0.0.0/24 addressing. However, packets forwarded from the home network into the larger global Internet clearly cannot use these addresses (as either a source or destination address) because there are hundreds of thousands of networks, each with its own 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 single 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 the use of 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 has a destination IP address of 138.76.29.7. In essence, the NAT-enabled router hides 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 assign addresses to computers within the NAT-DHCP-router-controlled home network address space.)

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 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 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 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 adds an entry to its NAT translation table. The Web server, blissfully unaware of 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 rather than addressing hosts. (This violation can indeed cause problems for servers on the home network, since, as we have seen in Chapter 2, server processes for incoming requests at well-known port numbers.) Second, they argue, the NAT protocol is not a standard Internet protocol; that is, hosts should not be able to communicate directly with each other, without interfering nodes modifying IP addresses and port numbers. And fourth, they argue, we should use IPv6 (see Section 4.6) 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

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 Peer A 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 media 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 in Skype 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 2012]. UPnP requires that both the host and the NAT be UPnP compatible. Using UPnP, an application running in a host can request a NAT mapping between (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 an internal 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 files 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 this SYN packet, it will change the destination IP address and port number of 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 2004] and [NAT 2012].

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 then calculated 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. 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. Specifically, 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 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 message). 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 a specified host. The destination host, seeing the echo request, sends back an ICMP type 0 code 0 ICMP echo reply. Most TCP/IP implementations support ping as a service directly in the operating system; that is, the server is not a process. Chapter 11 [Stevens 1990] provides the source code for the ping client program. Note that the ping 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, so the use of network-layer feedback such as the ICMP source quench message is not needed.

In Chapter 1 we introduced the Traceroute program, which allows us to determine the 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 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.

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 datagram. When the datagram arrives at the router, the 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 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; one of the datagrams will eventually make it all the way to the destination. 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 actually sends sets of three packets with the same TTL; thus the Traceroute provides three results for each TTL.)



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 system (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.

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 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 (so no new networks could attach to the Internet) was the subject of considerable debate. The estimates of the two leaders of the IETF’s Address Lifetime Extensions working group were that addresses would become exhausted in 2008 and 2011 respectively [Solensky 1996]. In February 2011, IANA allocated out the last remaining pool of unassigned IPv4 addresses to a regional registry. While these registries still have available IPv4 addresses within their pool, once these addresses are exhausted, there are no more available address blocks that can be allocated from a central pool [Huston 2011a]. Although the mid-1990s estimates of IPv4 address depletion 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 required 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] which we’ll discuss below. An often-asked question is what happened to IPv5? It was initially envisioned that the ST-2 protocol would become IPv5, but it was later dropped. Excellent sources of information about IPv6 are [Huitema 1998, IPv6 2012].

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 anycast address which allows a datagram to be delivered to 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 particular 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

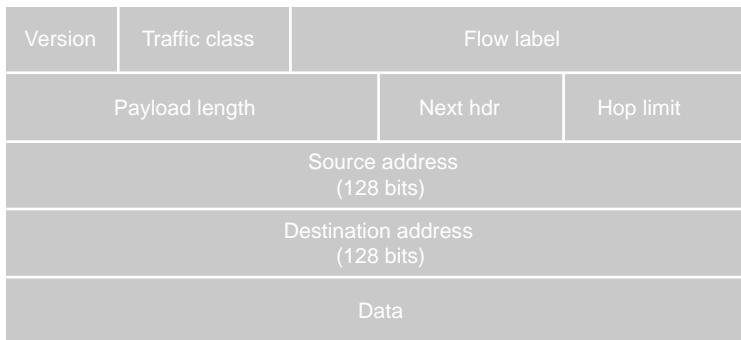


Figure 4.24 IPv6 datagram format

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

- € Flow labeling and priority IPv6 has an elusive definition of 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 of service or real-time service. Ž 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 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 between 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, news 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, it 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.)

- € Traffic class. This 8-bit field is similar in spirit to the TOS field we saw in IP.
- € 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 header.
- € Next header. This field identifies the protocol to which the contents (data payload) of this datagram will be delivered (for example, to TCP or UDP). The field contains 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 forwarding within the network.
- € Header checksum. Because the transport-layer (for example, TCP and UDP) and link-layer (for example, Ethernet) protocols in the Internet layers perform header checksumming, the designers of IP probably felt that this functionality was sufficiently redundant in the network layer that it could be removed. Once again, fairness in the processing of IP packets was a central concern. Recall from our discussion 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.

- € 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 headers pointed to from within the IPv6 header. That is, just as TCP or other protocol headers can be the next header within an IP packet, so too can the options field. The removal of the options field results in a fixed-length 64-byte IP header.

Recall from our discussion in Section 4.4.3 that the ICMP protocol is used by nodes to report error conditions and provide limited information (for example, an 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 types and code definitions, ICMPv6 also added new types and codes required by IPv6 functionality. These include the •Packet Too Big• type, and an •unreachable• 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 originally 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 transitioning to IPv6? The problem is that while new IPv6-capable systems can be made basically compatible, that is, can send, route, and receive IPv4 datagrams, already old IPv4-capable systems are not capable of handling IPv6 datagrams. Several approaches are possible [Huston 2011b].

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 first 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 alone or together) for gradually integrating IPv6 hosts and routers into a 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 the 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.

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

In the dual-stack approach, if either the sender or the receiver is only IPv6 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, as illustrated in Figure 4.25. Suppose Node A is IPv6-capable and wants to send an IPv6 datagram to Node F, which is also IPv6-capable. Nodes A and B can exchange IPv6 datagrams directly. However, Node B must create an IPv4 datagram to send to Node 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 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 of 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 data-

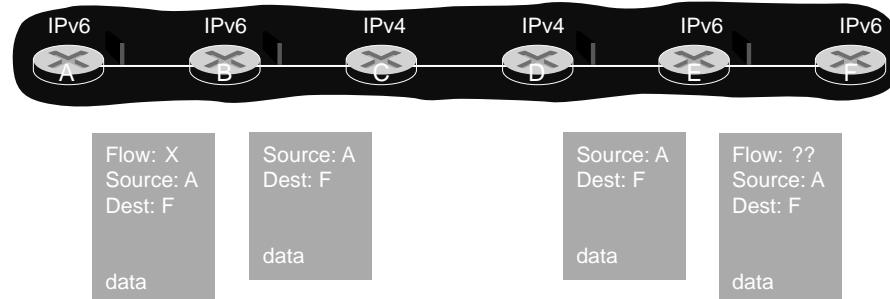


Figure 4.25 A dual-stack approach

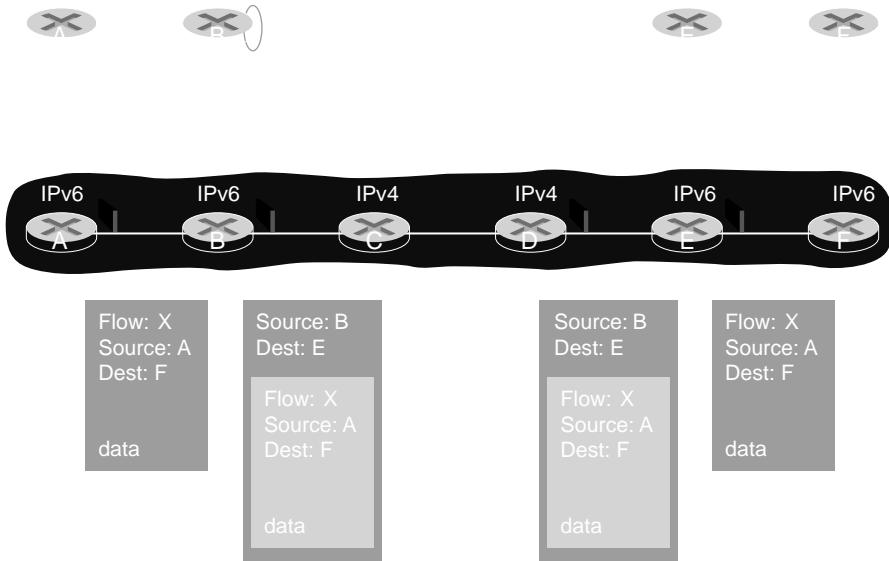


Figure 4.26 Tunneling

This IPv4 datagram is then addressed to the IPv6 node on the receiving side of the tunnel (for example, E) 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. See [Tonon 2008b] for discussion of IPv6 deployment as of 2008; see [NIST IPv6] for a snapshot of US IPv6 deployment. The proliferation of devices such as IPv6-enabled phones and other portable devices provides an additional push for IPv6.

widespread deployment of IPv6. Europe’s Third Generation Partnership Project [3GPP 2012] has specified IPv6 as the standard addressing scheme for multimedia.

One important lesson that we can learn from the IPv6 experience is that it is mously difficult to change network-layer protocols. Since the early 1990s, numerous new network-layer protocols have been trumpeted as the next major revolution of 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 longer 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 had 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’s cryptographic underpinnings are covered in some detail in Chapter 8, we provide a brief, high-level introduction into IPsec services in this section.

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 IP 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, they need 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

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, adds additional security fields to the segment, and encapsulates the resulting packet into an ordinary IP datagram. (It's actually a little more complicated than this, as we see in Chapter 8.) The sending host then sends the datagram into the Internet, where it transports it to the destination host. There, IPsec decrypts the segment and removes 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 data, including 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 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 laptop computer. Suppose the salespeople need to frequently consult sensitive company information (for example, pricing and product information) that is stored in a central server in the company's headquarters. Further suppose that the salespeople need to send sensitive documents to each other. How can this be done with IPsec? As you 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 central 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 routing 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

compute the information that is used to configure these forwarding tables. The play between routing algorithms and forwarding tables was shown in Figure 4.26. Having explored forwarding in some depth we now turn our attention to the 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 virtual circuit (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 travel 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 router 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 “route belonging to organization Y should not forward any packets originating from the network owned by organization X,” also come into play to complicate the conceptually simple and elegant algorithms whose design 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 connecting these nodes represent the physical links between these routers. A graph abstraction of a computer network is shown in Figure 4.27. To view graphs representing real network maps, see [Dodge 2012, Cheswick 2003]. A discussion of how well different graph-based models model the Internet [Zegura 1997, Faloutsos 1999, Li 2004].

As shown in Figure 4.27, an edge also has a value representing its cost. Closely, 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 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 .

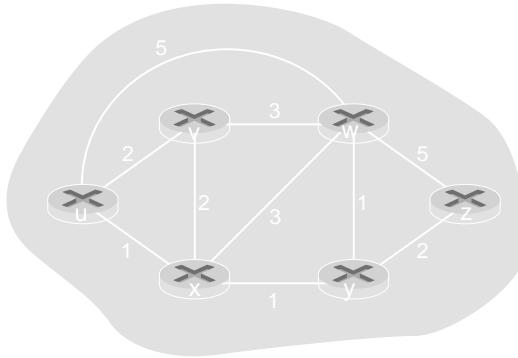


Figure 4.27 Abstract graph model of a computer network

Given that costs are assigned to the various edges in the graph abstractly, the goal of a routing algorithm is to identify the least costly paths between source and destination. To make this problem more precise, recall that in a graph $G = (N, E)$ is a sequence of nodes $x_1, (x_2, \dots, x_p)$ such that each of the pairs $(x_1, x_2), (x_2, x_3), \dots, (x_{p-1}, x_p)$ are edges in E . The cost of a path (x_1, \dots, x_p) is simply the sum of all the edge costs along the path, that is, $c(x_1, x_2) + c(x_2, x_3) + \dots + c(x_{p-1}, x_p)$. 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 with 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 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 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 categorize 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 input. This then requires that the algorithm somehow obtain this information before actually performing the calculation. The calculation itself can be run at one location.

(a centralized global routing algorithm) or replicated at multiple sites. The 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 knowledge of the costs of its own directly attached links. Then, through an iterative process of calculation and exchange of information with its neighbor nodes (that is, nodes that are at the other end of links to which it 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 changing a router's forwarding table). Dynamic routing algorithms change the routing paths as the network traffic loads or topology change. A dynamic algorithm can run either periodically or in direct response to topology or link cost changes. Dynamic algorithms are more responsive to network changes, they are also susceptible to problems such as routing loops and oscillation in routes.

A third way to classify routing algorithms is according to whether they are load-sensitive or load-insensitive. In load-sensitive algorithms, 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 all attached links. In practice (for example, with the Internet's OSPF routing protocol discussed in Section 4.6.1) this is often accomplished by link-state broadcasts.

algorithm [Perlman 1999]. We'll cover broadcast algorithms in Section 4.1. A 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 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 [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 i th iteration of the algorithm, the least-cost paths are known to destination nodes, and among the least-cost paths to all destination nodes, these paths will have the smallest costs. Let us define the following notation:

- € $D(v)$: cost of the least-cost path from the source node to destination v of this iteration of the algorithm.
- € $p(v)$: previous node (neighbor of v) along the current least-cost path from source to v .
- € N^* : subset of nodes v such that if the least-cost path from the source u 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.

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  $D(v) = \infty$ 
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 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 shows 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 its directly attached neighbors, 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-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 x has the least cost as of the end of the previous iteration. This is x, with a cost of 1, and thus 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 x has changed. 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 as the predecessor along the shortest path from 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 y and z are found to have the least-cost paths to u, and we break the tie arbitrarily and add 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 w, y, and z, 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,

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

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

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 4.27 shows the resulting least-cost paths and forwarding tables in 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 $n-1$ nodes to determine the node, not in N , that has the minimum cost. In the second iteration, we need to search through $n-2$ nodes to determine the minimum cost; in the third iteration 2 nodes, and so on. Overall, the total number of nodes we need to search through over all iterations is $n(n + 1)/2$, and thus we say that the preceding implementation of the LS algorithm has worst-case complexity of $O(n^2)$. (A more sophisticated implementation of this algorithm, using a data structure known as a min-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 topology 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 may be experienced. In this example, link costs are not symmetric; that is, $c(u,v) \neq c(v,u)$ only if the load carried on both directions on the link (u,v) is the same. In this example, node w originates a unit of traffic destined for node x , and node x also originates a unit of traffic destined for w , and node y injects an amount of traffic equal to α destined for w . The initial routing is shown in Figure 4.29(a) with the link costs responding to the amount of traffic carried.

When the LS algorithm is next run, node w determines (based on the link costs shown in Figure 4.29(a)) that the clockwise path $w \rightarrow v \rightarrow u \rightarrow w$ has a cost of 1, while the counter-clockwise path $w \rightarrow u \rightarrow v \rightarrow w$ (which it had been using) has a cost of $\alpha + 1$. Hence, w 's

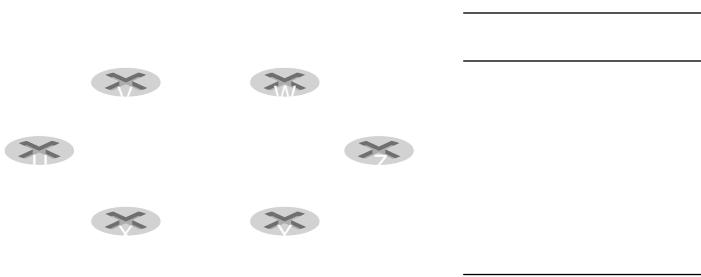
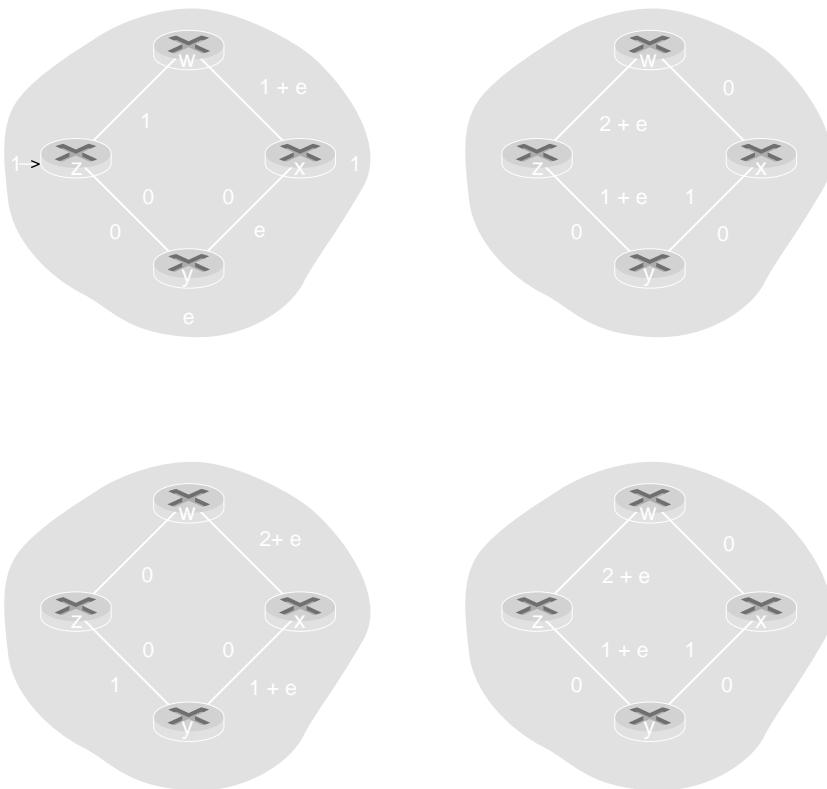


Figure 4.28 Least cost path and forwarding table for node u



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 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, 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 metric)? One solution would be to mandate that link costs not depend on the traffic carried, an unacceptable solution since one goal of routing is to a

This page intentionally left blank

We have just given a high-level overview of how an institution can use IPsec to create a VPN. To see the forest through the trees, we have brushed many important details. Let's now take a closer look.

8.7.2 The AH and ESP Protocols

IPsec is a rather complex animal, it is defined in more than a dozen RFCs. Two important RFCs are RFC 4301, which describes the overall IP security architecture, and RFC 6071, which provides an overview of the IPsec protocol suite. Our goal in this textbook, as usual, is not simply to re-hash the dry and arcane RFCs, but to take a more operational and pedagogic approach to describing the protocols.

In the IPsec protocol suite, there are two principal protocols: the Authentication Header (AH) protocol and the Encapsulation Security Payload (ESP) protocol. When a source IPsec entity (typically a host or a router) sends secure datagrams to a destination entity (also a host or a router), it does so with either the AH protocol or the ESP protocol. The AH protocol provides source authentication and data integrity, but does not provide confidentiality. The ESP protocol provides source authentication, data integrity, and confidentiality. Because confidentiality is often critical for VPNs and other IPsec applications, the ESP protocol is much more widely used than the AH protocol. In order to de-mystify IPsec and avoid much of its complication, we henceforth focus exclusively on the ESP protocol. Readers wanting to learn about the AH protocol are encouraged to explore the RFCs and other online resources.

8.7.3 Security Associations

IPsec datagrams are sent between pairs of network entities, such as between two hosts, between two routers, or between a host and router. Before sending IPsec datagrams from source entity to destination entity, the source and destination entities create a work-layer logical connection. This logical connection is called a security association (SA). An SA is a simplex logical connection; that is, it is unidirectional from source to destination. If both entities want to send secure datagrams to each other, then (that is, two logical connections) need to be established, one in each direction.

For example, consider once again the institutional VPN in Figure 8.27. This institution consists of a headquarters office, a branch office and traveling salespersons. For the sake of example, let's suppose that there is bi-directional IPsec traffic between headquarters and the branch office and bi-directional IPsec traffic between headquarters and the salespersons. In this VPN, how many SAs are there? To answer this question, note that there are two SAs between the headquarters gateway and the branch-office gateway router (one in each direction); for each salesperson, there are two SAs between the headquarters gateway router and the laptop (again in each direction). So, in total, there are $(2n+2)$ SAs. Keep in mind, however, that not all traffic sent into the Internet by the gateway routers or by the laptops will be secured. For example, a host in headquarters may want to access a Web server (such as Amazon or Google) in the public Internet. Thus, the gateway router (and the laptop) will emit into the Internet both vanilla IPv4 datagrams and secured IPsec datagrams.

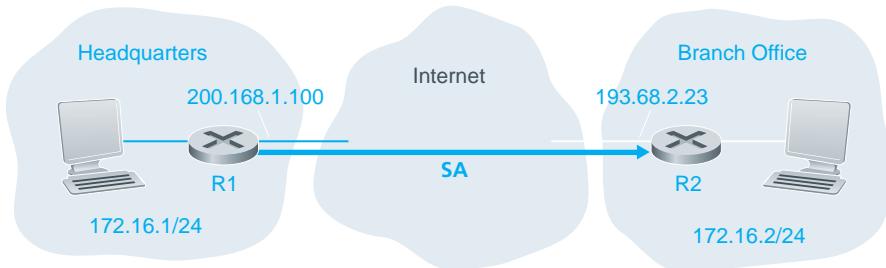


Figure 8.28 ♦ Security Association (SA) from R1 to R2

Let's now take a look inside an SA. To make the discussion tangible and concrete, let's do this in the context of an SA from router R1 to router R2 in Figure 8.28. (You can think of Router R1 as the headquarters gateway router and Router R2 as the branch office gateway router from Figure 8.27.) Router R1 will maintain state information about this SA, which will include:

- € A 32-bit identifier for the SA, called the **Security Parameter Index (SPI)**
- € The origin interface of the SA (in this case 200.168.1.100) and the destination interface of the SA (in this case 193.68.2.23)
- € The type of encryption to be used (for example, 3DES with CBC)
- € The encryption key
- € The type of integrity check (for example, HMAC with MD5)
- € The authentication key

Whenever router R1 needs to construct an IPsec datagram for forwarding over this SA, it accesses this state information to determine how it should authenticate and encrypt the datagram. Similarly, router R2 will maintain the same state information for this SA and will use this information to authenticate and decrypt any IPsec datagram that arrives from the SA.

An IPsec entity (router or host) often maintains state information for many SAs. For example, in the VPN example in Figure 8.27 with salespersons, the headquarters gateway router maintains state information for (2+2) SAs. An IPsec entity stores the state information for all of its SAs in **Security Association Database (SAD)**, which is a data structure in the entity's OS kernel.

8.7.4 The IPsec Datagram

Having now described SAs, we can now describe the actual IPsec datagram. IPsec has two different packet forms, one for the so-called **tunnel mode** and the other for the so-called **transport mode**. The tunnel mode, being more appropriate for VPNs, is more widely deployed than the transport mode. In order to further de-mystify

IPsec and avoid much of its complication, we henceforth focus exclusively on tunnel mode. Once you have a solid grip on the tunnel mode, you should be easily able to learn about the transport mode on your own.

The packet format of the IPsec datagram is shown in Figure 8.29. You might think that packet formats are boring and insipid, but we will soon see that the datagram actually looks and tastes like a popular Tex-Mex delicacy! Let's examine the IPsec fields in the context of Figure 8.28. Suppose router R1 receives an ordinary IPv4 datagram from host 172.16.1.17 (in the headquarters network) which is destined to host 172.16.2.48 (in the branch-office network). Router R1 uses the following recipe to convert this original IPv4 datagram into an IPsec datagram:

- € Appends to the back of the original IPv4 datagram (which includes the original IP header fields!) an “ESP trailer” field
- € Encrypts the result using the algorithm and key specified by the SA
- € Appends to the front of this encrypted quantity a field called “ESP header”. The resulting package is called the “enchilada”
- € Creates an authentication MAC over the whole enchilada using the algorithm and key specified in the SA
- € Appends the MAC to the back of the enchilada forming the “load”
- € Finally, creates a brand new IP header with all the classic IPv4 header fields (together normally 20 bytes long), which it appends before the payload

Note that the resulting IPsec datagram is a bona fide IPv4 datagram, with the traditional IPv4 header fields followed by a payload. But in this case, the payload contains an ESP header, the original IP datagram, an ESP trailer, and an ESP authentication field (with the original datagram and ESP trailer encrypted). The original IPv4 datagram has 172.16.1.17 for the source IP address and 172.16.2.48 for the destination IP address. Because the IPsec datagram includes the original IP datagram, the source and destination IP addresses are included (and encrypted) as part of the payload of the IPsec datagram. But what about the source and destination IP addresses that are in the new IP header? That is, in the left-most header of the IPsec datagram? As you might expect, these addresses are set to the source and destination router interfaces at the two ends of the link, namely, 200.168.1.100 and 193.68.2.23. Also, the protocol number in this new IP header field is not set to that of TCP, UDP, or SMTP, but instead to 50, designating that this is an IPsec datagram using the ESP protocol.

After R1 sends the IPsec datagram into the public Internet, it will pass through many routers before reaching R2. Each of these routers will process the data if it were an ordinary datagram, they are completely oblivious to the fact that the datagram is carrying IPsec-encrypted data. For these public Internet routers, the destination IP address in the outer header is R2, the ultimate destination of the datagram is R2.

Having walked through an example of how an IPsec datagram is constructed, let's now take a closer look at the ingredients in the enchilada. We see in Figure 8.29



Figure 8.29 ♦ IPsec datagram format

that the ESP trailer consists of three fields: padding; pad length; and next header. Recall that block ciphers require the message to be encrypted to be an integer multiple of the block length. Padding (consisting of meaningless bytes) is used so that when added to the original datagram (along with the pad length and next header fields), the resulting •message• is an integer number of blocks. The pad-length field indicates to the receiving entity how much padding was inserted (and thus needs to be removed). The next header identifies the type (e.g., UDP) of data contained in the payload-data field. The payload data (typically the original IP datagram) and the ESP trailer are concatenated and then encrypted.

Appended to the front of this encrypted unit is the ESP header, which is sent in the clear and consists of two fields: the SPI and the sequence number field. The SPI indicates to the receiving entity the SA to which the datagram belongs; the receiving entity can then index its SAD with the SPI to determine the appropriate authentication/decryption algorithms and keys. The sequence number field is used to defend against replay attacks.

The sending entity also appends an authentication MAC. As stated earlier, the sending entity calculates a MAC over the whole enchilada (consisting of the ESP header, the original IP datagram, and the ESP trailer, with the datagram and trailer being encrypted). Recall that to calculate a MAC, the sender appends a secret MAC key to the enchilada and then calculates a fixed-length hash of the result.

When R2 receives the IPsec datagram, R2 observes that the destination IP address of the datagram is R2 itself. R2 therefore processes the datagram. Because the protocol field (in the left-most IP header) is 50, R2 sees that it should apply IPsec ESP processing to the datagram. First, peering into the enchilada, R2 uses the SPI to determine to which SA the datagram belongs. Second, it calculates the MAC of the enchilada and verifies that the MAC is consistent with the value in the ESP MAC field. If it is, it knows that the enchilada comes from R1 and has not been tampered with. Third, it checks the sequence-number field to verify that the datagram is fresh (and not a replayed datagram). Fourth, it decrypts the encrypted unit using the

decryption algorithm and key associated with the SA. Fifth, it removes padding, extracts the original, vanilla IP datagram. And finally, sixth, it forwards the original datagram into the branch office network towards its ultimate destination. You know what a complicated recipe, huh? Well no one ever said that preparing and eating an enchilada was easy!

There is actually another important subtlety that needs to be addressed. Let's consider the following question: When R1 receives an (unsecured) datagram from some host in the headquarters network, and that datagram is destined to some host with a destination IP address outside of headquarters, how does R1 know whether it should convert the datagram to an IPsec datagram? And if it is to be processed by IPsec, how does R1 know which SA (of many SAs in its SAD) should be used to construct an IPsec datagram? The problem is solved as follows. Along with a SAD, the IPsec entity also maintains another data structure called the Security Policy Database (SPD). The SPD indicates what types of datagrams (as a function of source address, destination IP address, and protocol type) are to be IPsec processed, and which SA should be used. In a sense, the information in a SPD indicates •what• to do with an arriving datagram; the information in the SAD indicates •how• to do it.

Summary of IPsec Services

So what services does IPsec provide, exactly? Let us examine these services from the perspective of an attacker, say Trudy, who is a woman-in-the-middle, located somewhere on the path between R1 and R2 in Figure 8.28. Assume through discussion that Trudy does not know the authentication and encryption keys used in the SA. What can and cannot Trudy do? First, Trudy cannot see the original datagram. In fact, not only is the data in the original datagram hidden from Trudy, so is the protocol number, the source IP address, and the destination IP address of the datagrams sent over the SA. Trudy only knows that the datagram originated from some host in 172.16.1.0/24 and is destined to some host in 172.16.2.0/24. She does not know if it is carrying TCP, UDP, or ICMP data; she does not know if it is carrying HTTP, SMTP, or some other type of application data. This confidentiality goes a lot farther than SSL. Second, suppose Trudy tries to tamper with a datagram in the SA by flipping some of its bits. When this tampered datagram arrives at R2, it will fail the integrity check (using the MAC), thwarting Trudy's vicious attack once again. Third, suppose Trudy tries to masquerade as R1, creating a IPsec datagram with source 200.168.1.100 and destination 193.68.2.23. Trudy's attack is futile, as this datagram will again fail the integrity check at R2. Finally, because IPsec includes sequence numbers, Trudy will not be able to create a successful attack. In summary, as claimed at the beginning of this section, IPsec provides security between any pair of devices that process packets through the network, specifically guaranteeing confidentiality, source authentication, data integrity, and replay-attack prevention.

8.7.5 IKE: Key Management in IPsec

When a VPN has a small number of end points (for example, just two routers as in Figure 8.28), the network administrator can manually enter the SA information (encryption/authentication algorithms and keys, and the SPIs) into the SADs of the endpoints. Such manual keying is clearly impractical for a large VPN, which may consist of hundreds or even thousands of IPsec routers and hosts. Large, geographically distributed deployments require an automated mechanism for creating the SAs. IPsec does this with the Internet Key Exchange (IKE) protocol, specified in RFC 5996.

IKE has some similarities with the handshake in SSL (see Section 8.6). Each IPsec entity has a certificate, which includes the entity's public key. As with SSL, the IKE protocol has the two entities exchange certificates, negotiate authentication and encryption algorithms, and securely exchange key material for creating session keys in the IPsec SAs. Unlike SSL, IKE employs two phases to carry out these tasks.

Let's investigate these two phases in the context of two routers, R1 and R2, in Figure 8.28. The first phase consists of two exchanges of message pairs between R1 and R2:

- € During the first exchange of messages, the two sides use Diffie-Hellman (see Homework Problems) to create a bi-directional IKE SA between the routers. To keep us all confused, this bi-directional IKE SA is entirely different from the IPsec SAs discussed in Sections 8.6.3 and 8.6.4. The IKE SA provides an authenticated and encrypted channel between the two routers. During this first message-pair exchange, keys are established for encryption and authentication for the IKE SA. Also established is a master secret that will be used to compute IPsec SA keys later in phase 2. Observe that during this first step, RSA public and private keys are not used. In particular, neither R1 nor R2 reveals its identity by signing a message with its private key.
- € During the second exchange of messages, both sides reveal their identity to each other by signing their messages. However, the identities are not revealed to a passive sniffer, since the messages are sent over the secured IKE SA channel. Also during this phase, the two sides negotiate the IPsec encryption and authentication algorithms to be employed by the IPsec SAs.

In phase 2 of IKE, the two sides create an SA in each direction. At the end of phase 2, the encryption and authentication session keys are established on both sides for the two SAs. The two sides can then use the SAs to send secured datagrams, as described in Sections 8.7.3 and 8.7.4. The primary motivation for having two phases in IKE is computational cost, since the second phase doesn't involve any public-key cryptography, IKE can generate a large number of SAs between the two IPsec entities with relatively little computational cost.

8.8 Securing Wireless LANs

Security is a particularly important concern in wireless networks, where radio waves carrying frames can propagate far beyond the building containing the wireless base station and hosts. In this section we present a brief introduction to wireless security. For a more in-depth treatment, see the highly readable book by Edney and Arbaugh [Edney 2003].

The issue of security in 802.11 has attracted considerable attention in both technical circles and in the media. While there has been considerable discussion, there has been little debate, „there seems to be universal agreement that the original 802.11 specification contains a number of serious security flaws. Indeed, public domain software can now be downloaded that exploits these holes, making those who use the vanilla 802.11 security mechanisms as open to security attacks as users who use no security features at all.

In the following section, we discuss the security mechanisms initially standardized in the 802.11 specification, known collectively **Wired Equivalent Privacy (WEP)**. As the name suggests, WEP is meant to provide a level of security similar to that found in wired networks. We'll then discuss a few of the security holes in WEP and discuss the 802.11i standard, a fundamentally more secure version of 802.11 adopted in 2004.

8.8.1 Wired Equivalent Privacy (WEP)

The IEEE 802.11 WEP protocol was designed in 1999 to provide authentication and data encryption between a host and a wireless access point (that is, base station) using a symmetric shared key approach. WEP does not specify a key management algorithm, so it is assumed that the host and wireless access point have somehow agreed on the key via an out-of-band method. Authentication is carried out as follows:

1. A wireless host requests authentication by an access point.
2. The access point responds to the authentication request with a 128-byte nonce value.
3. The wireless host encrypts the nonce using the symmetric key that it shares with the access point.
4. The access point decrypts the host-encrypted nonce.

If the decrypted nonce matches the nonce value originally sent to the host, then the host is authenticated by the access point.

The WEP data encryption algorithm is illustrated in Figure 8.30. A secret 40-bit symmetric key K_S , is assumed to be known by both a host and the access point. In addition, a 24-bit Initialization Vector (IV) is appended to the 40-bit key to create a 64-bit key that will be used to encrypt a single frame. The IV will change from one

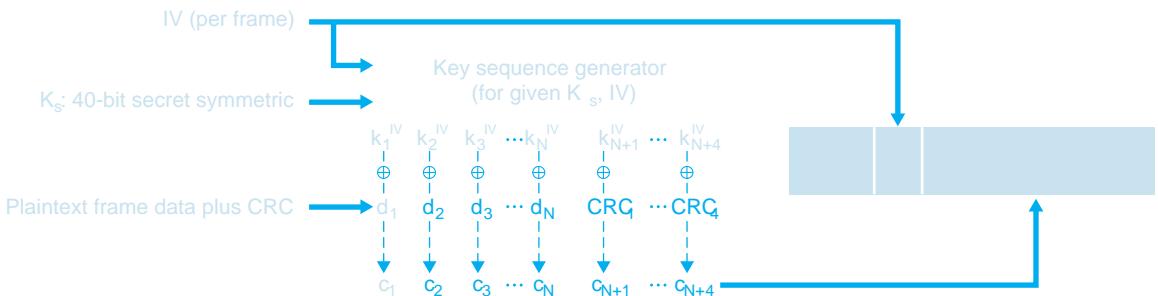


Figure 8.30 ♦ 802.11 WEP protocol

frame to another, and hence each frame will be encrypted with a different 64-bit key. Encryption is performed as follows. First a 4-byte CRC value (see Section 5.2) is computed for the data payload. The payload and the four CRC bytes are then encrypted using the RC4 stream cipher. We will not cover the details of RC4 here (see [Schneier 1995] and [Edney 2003] for details). For our purposes, it is enough to know that when presented with a key value (in this case, the 64-bit K_s key), the RC4 algorithm produces a stream of key values $k_1^{IV}, k_2^{IV}, k_3^{IV}, \dots$ that are used to encrypt the data and CRC value in a frame. For practical purposes, we can think of these operations being performed a byte at a time. Encryption is performed by XOR-ing the i th byte of data d_i , with the i th key, k_i^{IV} , in the stream of key values generated by the (K_s, IV) pair to produce the i th byte of ciphertext c_i :

$$c_i = d_i \oplus k_i^{IV}$$

The IV value changes from one frame to the next and is included in the header of each WEP-encrypted 802.11 frame, as shown in Figure 8.30. The receiver takes the secret 40-bit symmetric key that it shares with the sender, appends the IV, and uses the resulting 64-bit key (which is identical to the key used by the sender to perform encryption) to decrypt the frame:

$$d_i = c_i \oplus k_i^{IV}$$

Proper use of the RC4 algorithm requires that the same 64-bit key be used more than once. Recall that the WEP key changes on a frame-by-frame basis. For a given K_s (which changes rarely, if ever), this means that there are only 2^{24} unique keys. If these keys are chosen randomly, we can show [Walker 2000; Edney 2003] that the probability of having chosen the same IV value (and hence used the same 64-bit key) is more than 99 percent after only 12,000 frames. With 1 Kbyte frame sizes and a data transmission rate of 11 Mbps, only a few seconds are

needed before 12,000 frames are transmitted. Furthermore, since the IV is transmitted in plaintext in the frame, an eavesdropper will know whenever a duplicate IV value is used.

To see one of the several problems that occur when a duplicate key is used, consider the following chosen-plaintext attack taken by Trudy against Alice. Suppose that Trudy (possibly using IP spoofing) sends a request (for example, an HTTP or FTP request) to Alice to transmit a file with known content $d_1, d_2, d_3, d_4, \dots$. Trudy also observes the encrypted data $c_1, c_2, c_3, c_4, \dots$. Since $d_i = c_i \oplus k_i^{IV}$, if we XOR c_i with each side of this equality we have

$$d_i \oplus c_i = k_i^{IV}$$

With this relationship, Trudy can use the known values d_i and c_i to compute k_i^{IV} . The next time Trudy sees the same value of IV being used, she will know the key sequence $k_1^{IV}, k_2^{IV}, k_3^{IV}, \dots$ and will thus be able to decrypt the encrypted message.

There are several additional security concerns with WEP as well. [Fluhrer 2001] described an attack exploiting a known weakness in RC4 when certain weak keys are chosen. [Stubblefield 2002] discusses efficient ways to implement and exploit this attack. Another concern with WEP involves the CRC bits shown in Figure 8.30 and transmitted in the 802.11 frame to detect altered bits in the payload. However, an attacker who changes the encrypted content (e.g., substituting gibberish for the original encrypted data), computes a CRC over the substituted gibberish, and places the CRC into a WEP frame can produce an 802.11 frame that will be accepted by the receiver. What is needed here are message integrity techniques such as those we studied in Section 8.3 to detect content tampering or substitution. For more details of WEP security, see [Edney 2003; Walker 2000; Weatherspoon 2000] and the references therein.

8.8.2 IEEE 802.11i

Soon after the 1999 release of IEEE 802.11, work began on developing a new and improved version of 802.11 with stronger security mechanisms. The new standard, known as 802.11i, underwent final ratification in 2004. As we'll see, while WEP provided relatively weak encryption, only a single way to perform authentication, and no key distribution mechanisms, IEEE 802.11i provides for much stronger forms of encryption, an extensible set of authentication mechanisms, and a key distribution mechanism. In the following, we present an overview of 802.11i; an excellent (streaming audio) technical overview of 802.11i is [TechOnline 2012].

Figure 8.31 overviews the 802.11i framework. In addition to the wireless client and access point, 802.11i defines an authentication server with which the AP can communicate. Separating the authentication server from the AP allows one authentication server to serve many APs, centralizing the (often sensitive) decisions

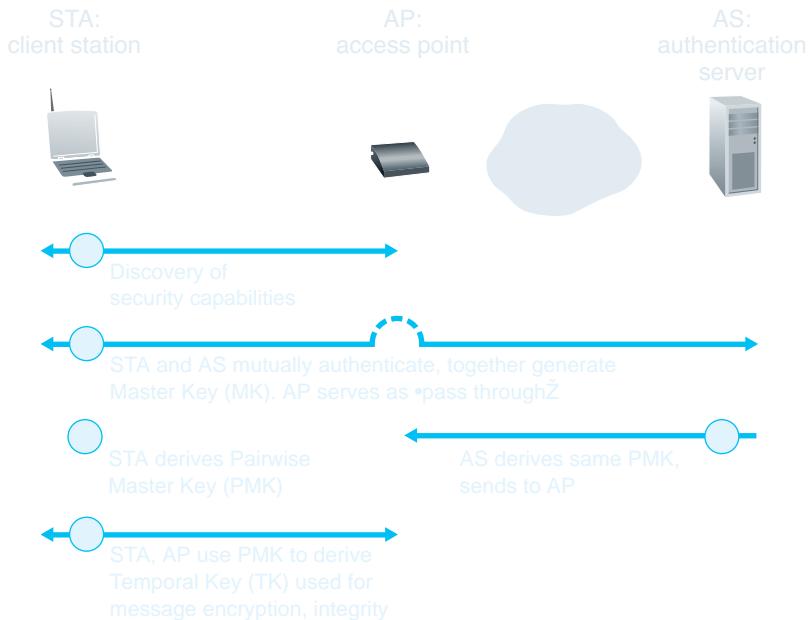


Figure 8.31 ♦ 802.11i: four phases of operation

regarding authentication and access within the single server, and keeping AP costs and complexity low. 802.11i operates in four phases:

1. *Discovery.* In the discovery phase, the AP advertises its presence and the forms of authentication and encryption that can be provided to the wireless client node. The client then requests the specific forms of authentication and encryption that it desires. Although the client and AP are already exchanging messages, the client has not yet been authenticated nor does it have an encryption key, and so several more steps will be required before the client can communicate with an arbitrary remote host over the wireless channel.
2. *Mutual authentication and Master Key (MK) generation.* Authentication takes place between the wireless client and the authentication server. In this phase, the access point acts essentially as a relay, forwarding messages between the client and the authentication server. **Extensible Authentication Protocol (EAP)** [RFC 3748] defines the end-to-end message formats used in a simple request/response mode of interaction between the client and authentication server. As shown in Figure 8.32 EAP messages are encapsulated using **EAPoL** (EAP over LAN, [IEEE 802.1X]) and sent over the 802.11 wireless link. These EAP messages are then decapsulated at the access point, and then

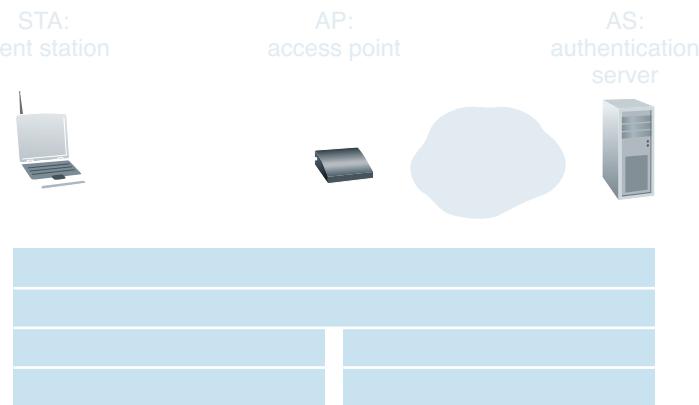


Figure 8.32 ♦ EAP is an end-to-end protocol. EAP messages are encapsulated using EAPoL over the wireless link between the client and the access point, and using RADIUS over UDP/IP between the access point and the authentication server

re-encapsulated using the RADIUS protocol for transmission over UDP/IP to the authentication server. While the RADIUS server and protocol [RFC 2865] are not required by the 802.11i protocol, they *are facto* standard components for 802.11i. The recently standardized DIAMETER protocol [RFC 3588] is unlikely to replace RADIUS in the near future.

With EAP, the authentication server can choose one of a number of ways to perform authentication. While 802.11i does not mandate a particular authentication method, the EAP-TLS authentication scheme [RFC 5216] is often used. EAP-TLS uses public key techniques (including nonce encryption and message digests) similar to those we studied in Section 8.3 to allow the client and the authentication server to mutually authenticate each other, and to derive a Master Key (MK) that is known to both parties.

3. *Pairwise Master Key (PMK) generation.* The MK is a shared secret known only to the client and the authentication server, which they each use to generate a second key, the Pairwise Master Key (PMK). The authentication server then sends the PMK to the AP. This is where we wanted to be! The client and AP now have a shared key (recall that in WEP, the problem of key distribution was not addressed at all) and have mutually authenticated each other. They're just about ready to get down to business.
4. *Temporal Key (TK) generation.* With the PMK, the wireless client and AP can now generate additional keys that will be used for communication. Of particular interest is the Temporal Key (TK), which will be used to perform the link-level encryption of data sent over the wireless link and to an arbitrary remote host.

802.11i provides several forms of encryption, including an AES-based encryption scheme and a strengthened version of WEP encryption.

8.9 Operational Security: Firewalls and Intrusion Detection Systems

We've seen throughout this chapter that the Internet is not a very safe place. Bad guys are out there, wreaking all sorts of havoc. Given the hostile nature of the Internet, let's now consider an organization's network and the network administrator who administers it. From a network administrator's point of view, the world divides quite neatly into two camps, the good guys (who belong to the organization's network, and who should be able to access resources inside the organization's network in a relatively unconstrained manner) and the bad guys (everyone else, whose access to network resources must be carefully scrutinized). In organizations, ranging from medieval castles to modern corporate office buildings, there is a single point of entry/exit where both good guys and bad guys enter and leave the organization are security-checked. In a castle, this was done at the drawbridge gate at one end of the drawbridge; in a corporate building, this is done at the security desk. In a computer network, when traffic entering/leaving a network is security-checked, logged, dropped, or forwarded, it is done by operational devices known as firewalls, intrusion detection systems (IDSs), and intrusion prevention systems (IPSs).

8.9.1 Firewalls

A firewall is a combination of hardware and software that isolates an organization's internal network from the Internet at large, allowing some packets to pass and blocking others. A firewall allows a network administrator to control access between the outside world and resources within the administered network by managing the traffic flow to and from these resources. A firewall has three goals:

- € All traffic from outside to inside, and vice versa, passes through the firewall. Figure 8.33 shows a firewall, sitting squarely at the boundary between the administered network and the rest of the Internet. While large organizations may have multiple levels of firewalls or distributed firewalls [Skoudis 2006], locating a firewall at a single access point to the network, as shown in Figure 8.33, makes it easier to manage and enforce a security-access policy.
- € Only authorized traffic, as defined by the local security policy, will be allowed to pass. With all traffic entering and leaving the institutional network passing through the firewall, the firewall can restrict access to authorized traffic.

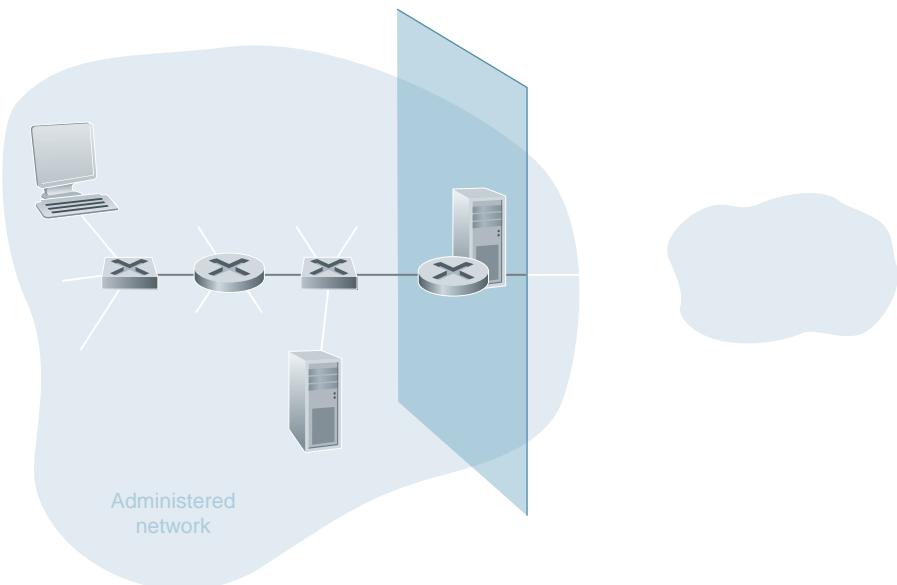


Figure 8.33 ♦ Firewall placement between the administered network and the outside world

€ The firewall itself is immune to penetration. The firewall itself is a device connected to the network. If not designed or installed properly, it can be compromised, in which case it provides only a false sense of security (which is worse than no firewall at all!).

Cisco and Check Point are two of the leading firewall vendors today. You can also easily create a firewall (packet filter) from a Linux box using iptables (public-domain software that is normally shipped with Linux).

Firewalls can be classified in three categories: **traditional packet filters**, **stateful filters**, and **application gateways**. We'll cover each of these in turn in the following subsections.

Traditional Packet Filters

As shown in Figure 8.33, an organization typically has a gateway router connecting its internal network to its ISP (and hence to the larger public Internet). All traffic leaving and entering the internal network passes through this router, and it is at this router where **packet filtering** occurs. A packet filter examines each datagram in isolation, determining whether the datagram should be allowed to pass or should be dropped based on administrator-specific rules. Filtering decisions are typically based on:

- € IP source or destination address
- € Protocol type in IP datagram field: TCP, UDP, ICMP, OSPF, and so on
- € TCP or UDP source and destination port
- € TCP flag bits: SYN, ACK, and so on
- € ICMP message type
- € Different rules for datagrams leaving and entering the network
- € Different rules for the different router interfaces

A network administrator configures the firewall based on the policy of the organization. The policy may take user productivity and bandwidth usage into account as well as the security concerns of an organization. Table 8.5 lists a number of possible policies an organization may have, and how they would be addressed with a packet filter. For example, if the organization doesn't want any incoming TCP connections except those for its public Web server, it can block all incoming TCP SYN segments except TCP SYN segments with destination port 80 and the destination IP address corresponding to the Web server. If the organization doesn't want its users to monopolize access bandwidth with Internet radio applications, it can block all not-critical UDP traffic (since Internet radio is often sent over UDP). If the organization doesn't want its internal network to be mapped (tracerouted) by an outsider, it can block all ICMP TTL expired messages leaving the organization's network.

A filtering policy can be based on a combination of addresses and port numbers. For example, a filtering router could forward all Telnet datagrams (those with a port number of 23) except those going to and coming from a list of specific IP addresses. This policy permits Telnet connections to and from hosts on the allowed

Policy	Firewall Setting
No outside Web access.	Drop all outgoing packets to any IP address, port 80
No incoming TCP connections, except those for organization's public Web server only.	Drop all incoming TCP SYN packets to any IP except 130.207.244.203, port 80
Prevent Web radios from eating up the available bandwidth.	Drop all incoming UDP packets—except DNS packets.
Prevent your network from being used for a smurf DoS attack.	Drop all ICMP ping packets going to a "broadcast" address (eg 130.207.255.255).
Prevent your network from being tracerouted	Drop all outgoing ICMP TTL expired traffic

Table 8.5 ♦ Policies and corresponding filtering rules for an organization's network 130.27.16 with Web server at 130.207.244.203

list. Unfortunately, basing the policy on external addresses provides no protection against datagrams that have had their source addresses spoofed.

Filtering can also be based on whether or not the TCP ACK bit is set. This trick is quite useful if an organization wants to let its internal clients connect to external servers but wants to prevent external clients from connecting to internal servers. Recall from Section 3.5 that the first segment in every TCP connection has the ACK bit set to 0, whereas all the other segments in the connection have the ACK bit set to 1. Thus, if an organization wants to prevent external clients from initiating connections to internal servers, it simply filters all incoming segments with the ACK bit set to 0. This policy kills all TCP connections originating from the outside, but permits connections originating internally.

Firewall rules are implemented in routers with access control lists, with each router interface having its own list. An example of an access control list for an organization 222.22/16 is shown in Table 8.6. This access control list is for an interface that connects the router to the organization’s external ISPs. Rules are applied to each datagram that passes through the interface from top to bottom. The first two rules together allow internal users to surf the Web: The first rule allows any TCP packet with destination port 80 to leave the organization’s network; the second rule allows any TCP packet with source port 80 and the ACK bit set to enter the organization’s network. Note that if an external source attempts to establish a TCP connection with an internal host, the connection will be blocked, even if the source or destination port is 80. The second two rules together allow DNS packets to enter and leave the organization’s network. In summary, this rather restrictive access control list blocks all traffic except Web traffic initiated from within the organization and DNS traffic. [CERT Filtering 2012] provides a list of recommended port/protocol packet filterings to avoid a number of well-known security holes in existing network applications.

action	source address	dest address	protocol	source port	dest port	flag bit
allow	222.22/16	outside of 222.22/16	TCP	> 1023	80	any
allow	outside of 222.22/16	222.22/16	TCP	80	> 1023	ACK
allow	222.22/16	outside of 222.22/16	UDP	> 1023	53	—
allow	outside of 222.22/16	222.22/16	UDP	53	> 1023	—
deny	all	all	all	all	all	all

Table 8.6 ◆ An access control list for a router interface

Stateful Packet Filters

In a traditional packet filter, filtering decisions are made on each packet in isolation. Stateful filters actually track TCP connections, and use this knowledge to make filtering decisions.

To understand stateful filters, let's reexamine the access control list in Table 8.6. Although rather restrictive, the access control list in Table 8.6 nevertheless allows any packet arriving from the outside with ACK = 1 and source port 80 to get through the filter. Such packets could be used by attackers in attempts to crash internal systems with malformed packets, carry out denial-of-service attacks, or map the internal network. The naive solution is to block TCP ACK packets as well, but such an approach would prevent the organization's internal users from surfing the Web.

Stateful filters solve this problem by tracking all ongoing TCP connections in a connection table. This is possible because the firewall can observe the beginning of a new connection by observing a three-way handshake (SYN, SYNACK, and ACK); and it can observe the end of a connection when it sees a FIN packet for the connection. The firewall can also (conservatively) assume that the connection is over when it hasn't seen any activity over the connection for, say, 60 seconds. An example connection table for a firewall is shown in Table 8.7. This connection table indicates that there are currently three ongoing TCP connections, all of which have been initiated from within the organization. Additionally, the stateful filter includes a new column, "check connection," in its access control list, as shown in Table 8.8. Note that Table 8.8 is identical to the access control list in Table 8.6, except now it indicates that the connection should be checked for two of the rules.

Let's walk through some examples to see how the connection table and the extended access control list work hand-in-hand. Suppose an attacker attempts to send a malformed packet into the organization's network by sending a datagram with TCP source port 80 and with the ACK flag set. Further suppose that this packet has source port number 12543 and source IP address 150.23.23.155. When this packet reaches the firewall, the firewall checks the access control list in Table 8.7, which indicates that the connection table must also be checked before permitting this packet to enter the organization's network. The firewall duly checks the connection table, sees that this packet is not part of an ongoing TCP connection, and rejects

source address	dest address	source port	dest port
222.22.1.7	37.96.87.123	12699	80
222.22.93.2	199.1.205.23	37654	80
222.22.65.143	203.77.240.43	48712	80

Table 8.7 ♦ Connection table for stateful filter

action	source address	dest address	protocol	source port	dest port	flag bit	check connxion
allow	222.22/16	outside of 222.22/16	TCP	>1023	80	any	
allow	outside of 222.22/16	222.22/16	TCP	80	>1023	ACK	X
allow	222.22/16	outside of 222.22/16	UDP	>1023	53	—	
allow	outside of 222.22/16	222.22/16	UDP	53	>1023	—	X
deny	all	all	all	all	all	all	

Table 8.8 ♦ Access control list for stateful filter

the packet. As a second example, suppose that an internal user wants to surf an external Web site. Because this user first sends a TCP SYN segment, the user's TCP connection gets recorded in the connection table. When the Web server sends back packets (with the ACK bit necessarily set), the firewall checks the table and sees that a corresponding connection is in progress. The firewall will thus let these packets pass, thereby not interfering with the internal user's Web surfing activity.

Application Gateway

In the examples above, we have seen that packet-level filtering allows an organization to perform coarse-grain filtering on the basis of the contents of IP and TCP/UDP headers, including IP addresses, port numbers, and acknowledgment bits. But what if an organization wants to provide a Telnet service to a restricted set of internal users (as opposed to IP addresses)? And what if the organization wants such privileged users to authenticate themselves first before being allowed to create Telnet sessions to the outside world? Such tasks are beyond the capabilities of traditional and stateful filters. Indeed, information about the identity of the internal users is application-layer data and is not included in the IP/TCP/UDP headers.

To have finer-level security, firewalls must combine packet filters with application gateways. Application gateways look beyond the IP/TCP/UDP headers and make policy decisions based on application data. An application gateway is an application-specific server through which all application data (inbound and outbound) must pass. Multiple application gateways can run on the same host, but each gateway is a separate server with its own processes.

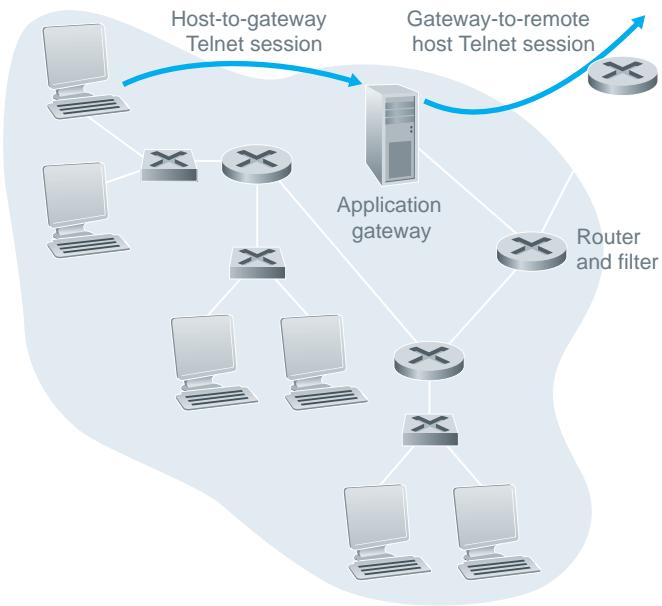


Figure 8.34 ♦ Firewall consisting of an application gateway and a filter

To get some insight into application gateways, let's design a firewall that allows only a restricted set of internal users to Telnet outside and prevents all external clients from Telneting inside. Such a policy can be accomplished by implementing a combination of a packet filter (in a router) and a Telnet application gateway, as shown in Figure 8.34. The router's filter is configured to block all Telnet connections except those that originate from the IP address of the application gateway. Such a filter configuration forces all outbound Telnet connections to pass through the application gateway. Consider now an internal user who wants to Telnet to the outside world. The user must first set up a Telnet session with the application gateway. An application running in the gateway, which listens for incoming Telnet sessions, prompts the user for a user ID and password. When the user supplies this information, the application gateway checks to see if the user has permission to Telnet to the outside world. If not, the Telnet connection from the internal user to the gateway is terminated by the gateway. If the user has permission, then the gateway (1) prompts the user for the host name of the external host to which the user wants to connect, (2) sets up a Telnet session between the gateway and the external host, and (3) relays to the external host all data arriving from the user, and relays to the user all data arriving from the external host. Thus, the Telnet application gateway not only performs user authorization but also acts as a Telnet server and a Telnet client, relaying information between the user and the remote Telnet server. Note that

the filter will permit step 2 because the gateway initiates the Telnet connection to the outside world.



CASE HISTORY

ANONYMITY AND PRIVACY

Suppose you want to visit a controversial Web site (for example, a political activist site) and you (1) don't want to reveal your IP address to the Web site, (2) don't want your local ISP (which may be your home or office ISP) to know that you are visiting the site, and (3) you don't want your local ISP to see the data you are exchanging with the site. If you use the traditional approach of connecting directly to the Web site without any encryption, you fail on all three counts. Even if you use SSL, you fail on the first two counts: Your source IP address is presented to the Web site in every datagram you send; and the destination address of every packet you send can easily be sniffed by your local ISP.

To obtain privacy and anonymity, you can instead use a combination of a trusted proxy server and SSL, as shown in Figure 8.35. With this approach, you first make an SSL connection to the trusted proxy. You then send, into this SSL connection, an HTTP request for a page at the desired site. When the proxy receives the SSL-encrypted HTTP request, it decrypts the request and forwards the cleartext HTTP request to the Web site. The Web site then responds to the proxy, which in turn forwards the response to you over SSL. Because the Web site only sees the IP address of the proxy, and not of your client's address, you are indeed obtaining anonymous access to the Web site. And because all traffic between you and the proxy is encrypted, your local ISP cannot invade your privacy by logging the site you visited or recording the data you are exchanging. Many companies today (such as proxify.com) make available such proxy services.

Of course, in this solution, your proxy knows everything: It knows your IP address and the IP address of the site you're surfing; and it can see all the traffic in cleartext exchanged between you and the Web site. Such a solution, therefore, is only as good as the trustworthiness of the proxy. A more robust approach, taken by the TOR anonymizing and privacy service, is to route your traffic through a series of non-colluding proxy servers [TOR 2012]. In particular, TOR allows independent individuals to contribute proxies to its proxy pool. When a user connects to a server using TOR, TOR randomly chooses (from its proxy pool) a chain of three proxies and routes all traffic between client and server over the chain. In this manner, assuming the proxies do not collude, no one knows that communication took place between your IP address and the target Web site. Furthermore, although cleartext is sent between the last proxy and the server, the last proxy doesn't know what IP address is sending and receiving the cleartext.

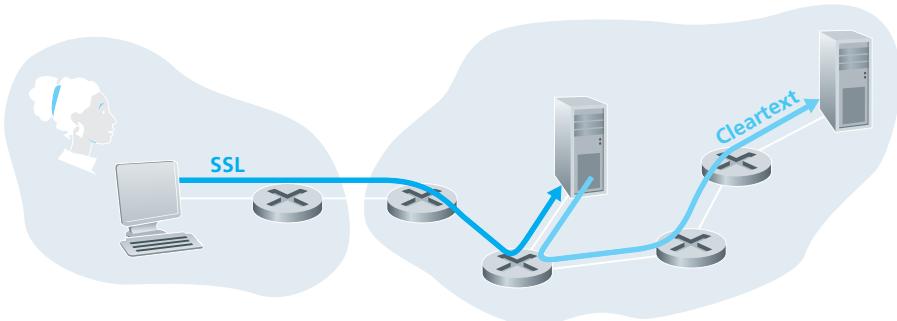


Figure 8.35 ♦ Providing anonymity and privacy with a proxy

Internal networks often have multiple application gateways, for example, gateways for Telnet, HTTP, FTP, and e-mail. In fact, an organization’s mail server (see Section 2.4) and Web cache are application gateways.

Application gateways do not come without their disadvantages. First, a different application gateway is needed for each application. Second, there is a performance penalty to be paid, since all data will be relayed via the gateway. This becomes a concern particularly when multiple users or applications are using the same gateway machine. Finally, the client software must know how to contact the gateway when the user makes a request, and must know how to tell the application gateway what external server to connect to.

8.9.2 Intrusion Detection Systems

We've just seen that a packet filter (traditional and stateful) inspects IP, TCP, UDP, and ICMP header fields when deciding which packets to let pass through the firewall. However, to detect many attack types, we need to perform **deep packet inspection**, that is, look beyond the header fields and into the actual application data that the packets carry. As we saw in Section 8.9.1, application gateways often do deep packet inspection. But an application gateway only does this for a specific application.

Clearly, there is a niche for yet another device, a device that not only examines the headers of all packets passing through it (like a packet filter), but also performs deep packet inspection (unlike a packet filter). When such a device observes a suspicious packet, or a suspicious series of packets, it could prevent those packets from entering the organizational network. Or, because the activity is only deemed as suspicious, the device could let the packets pass, but send alerts to a network administrator, who can then take a closer look at the traffic and take appropriate

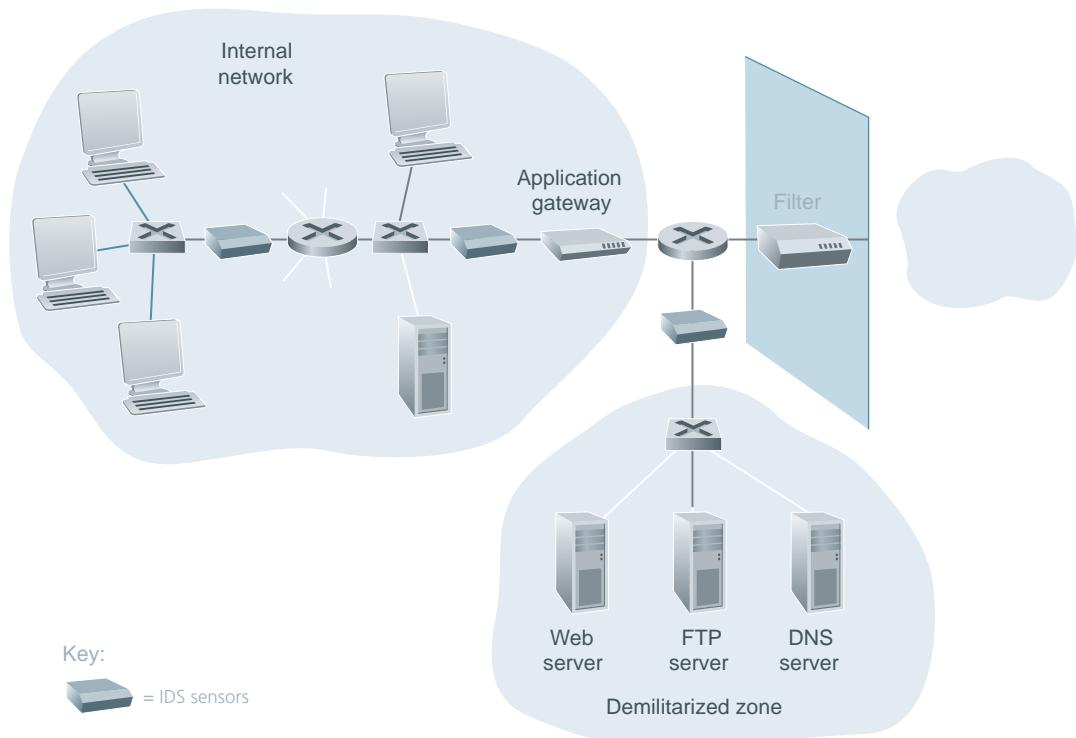


Figure 8.36 ♦ An organization deploying a filter, an application gateway, and IDS sensors

actions. A device that generates alerts when it observes potentially malicious traffic is called an **intrusion detection system (IDS)**. A device that filters out suspicious traffic is called an **intrusion prevention system (IPS)**. In this section we study both systems, IDS and IPS, together, since the most interesting technical aspect of these systems is how they detect suspicious traffic (and not whether they send alerts or drop packets). We will henceforth collectively refer to IDS systems and IPS systems as **IDS systems**.

An IDS can be used to detect a wide range of attacks, including network mapping (emanating, for example, from nmap), port scans, TCP stack scans, DoS bandwidth-flooding attacks, worms and viruses, OS vulnerability attacks, and application vulnerability attacks. (See Section 1.6 for a survey of network attacks.) Today, thousands of organizations employ IDS systems. Many of these deployed systems are proprietary, marketed by Cisco, Check Point, and other security equipment vendors. But many of the deployed IDS systems are public-domain systems, such as the immensely popular Snort IDS system (which we'll discuss shortly).

An organization may deploy one or more IDS sensors in its organizational network. Figure 8.36 shows an organization that has three IDS sensors. When multiple sensors are deployed, they typically work in concert, sending information about suspicious traffic activity to a central IDS processor, which collects and integrates the information and sends alarms to network administrators when deemed appropriate. In Figure 8.36, the organization has partitioned its network into two regions: a high-security region protected by a packet filter and an application gateway and monitored by IDS sensors; and a lower-security region, referred to as the ~~militarized zone (DMZ)~~, which is protected only by the packet filter, but also monitored by IDS sensors. Note that the DMZ includes the organization's servers that need to communicate with the outside world, such as its public Web server and its authoritative DNS server.

You may be wondering at this stage, why multiple IDS sensors? Why not just place one IDS sensor just behind the packet filter (or even integrated with the packet filter) in Figure 8.36? We will soon see that an IDS not only needs to do deep packet inspection, but must also compare each passing packet with tens of thousands of signatures; this can be a significant amount of processing, particularly if the organization receives gigabits/sec of traffic from the Internet. By placing the IDS sensors further downstream, each sensor sees only a fraction of the organization's traffic, and can more easily keep up. Nevertheless, high-performance IDS and IPS systems are available today, and many organizations can actually get by with just one sensor located near its access router.

IDS systems are broadly classified as either **signature-based systems** or **anomaly-based systems**. A signature-based IDS maintains an extensive database of attack signatures. Each signature is a set of rules pertaining to an intrusion activity. A signature may simply be a list of characteristics about a single packet (e.g., source and destination port numbers, protocol type, and a specific string of bits in the packet payload), or may relate to a series of packets. The signatures are normally created by skilled network security engineers who research known attacks. An organization's network administrator can customize the signatures or add its own to the database.

Operationally, a signature-based IDS sniffs every packet passing by it, comparing each sniffed packet with the signatures in its database. If a packet (or series of packets) matches a signature in the database, the IDS generates an alert. The alert could be sent to the network administrator in an e-mail message, could be sent to the network management system, or could simply be logged for future inspection.

Signature-based IDS systems, although widely deployed, have a number of limitations. Most importantly, they require previous knowledge of the attack to generate an accurate signature. In other words, a signature-based IDS is completely blind to new attacks that have yet to be recorded. Another disadvantage is that even if a signature is matched, it may not be the result of an attack, so that a false alarm is generated. Finally, because every packet must be compared with an extensive collection of signatures, the IDS can become overwhelmed with processing and actually fail to detect many malicious packets.

An anomaly-based IDS creates a traffic profile as it observes traffic in normal operation. It then looks for packet streams that are statistically unusual, for example, an inordinate percentage of ICMP packets or a sudden exponential growth in port scans and ping sweeps. The great thing about anomaly-based IDS systems is that they don't rely on previous knowledge about existing attacks, that is, they can potentially detect new, undocumented attacks. On the other hand, it is an extremely challenging problem to distinguish between normal traffic and statistically unusual traffic. To date, most IDS deployments are primarily signature-based, although some include some anomaly-based features.

Snort

Snort is a public-domain, open source IDS with hundreds of thousands of existing deployments [Snort 2012; Koziol 2003]. It can run on Linux, UNIX, and Windows platforms. It uses the generic sniffing interface libpcap, which is also used by Wireshark and many other packet sniffers. It can easily handle 100 Mbps of traffic; for installations with gigabit/sec traffic rates, multiple Snort sensors may be needed.

To gain some insight into Snort, let's take a look at an example of a Snort signature:

```
alert icmp $EXTERNAL_NET any -> $HOME_NET any  
(msg:"ICMP PING NMAP"; dsiz: 0; itype: 8;)
```

This signature is matched by any ICMP packet that enters the organization's network (\$HOME_NET) from the outside (\$EXTERNAL_NET), is of type 8 (ICMP ping), and has an empty payload (dsiz = 0). Since nmap (see Section 1.6) generates ping packets with these specific characteristics, this signature is designed to detect nmap ping sweeps. When a packet matches this signature, Snort generates an alert that includes the message "ICMP PING NMAP".

Perhaps what is most impressive about Snort is the vast community of users and security experts that maintain its signature database. Typically within a few hours of a new attack, the Snort community writes and releases an attack signature, which is then downloaded by the hundreds of thousands of Snort deployments distributed around the world. Moreover, using the Snort signature syntax, network administrators can tailor the signatures to their own organization's needs by either modifying existing signatures or creating entirely new ones.

8.10 Summary

In this chapter, we've examined the various mechanisms that our secret lovers, Bob and Alice, can use to communicate securely. We've seen that Bob and Alice are interested in confidentiality (so they alone are able to understand the contents of a transmitted message), end-point authentication (so they are sure that they are talking

with each other), and message integrity (so they are sure that their messages are not altered in transit). Of course, the need for secure communication is not confined to secret lovers. Indeed, we saw in Sections 8.5 through 8.8 that security can be used in various layers in a network architecture to protect against bad guys who have a large arsenal of possible attacks at hand.

The first part of this chapter presented various principles underlying secure communication. In Section 8.2, we covered cryptographic techniques for encrypting and decrypting data, including symmetric key cryptography and public key cryptography. DES and RSA were examined as specific case studies of these two major classes of cryptographic techniques in use in today's networks.

In Section 8.3, we examined two approaches for providing message integrity: message authentication codes (MACs) and digital signatures. The two approaches have a number of parallels. Both use cryptographic hash functions and both techniques enable us to verify the source of the message as well as the integrity of the message itself. One important difference is that MACs do not rely on encryption whereas digital signatures require a public key infrastructure. Both techniques are extensively used in practice, as we saw in Sections 8.5 through 8.8. Furthermore, digital signatures are used to create digital certificates, which are important for verifying the validity of public keys. In Section 8.4, we examined endpoint authentication and introduced nonces to defend against the replay attack.

In Sections 8.5 through 8.8 we examined several security networking protocols that enjoy extensive use in practice. We saw that symmetric key cryptography is at the core of PGP, SSL, IPsec, and wireless security. We saw that public key cryptography is crucial for both PGP and SSL. We saw that PGP uses digital signatures for message integrity, whereas SSL and IPsec use MACs. Having now an understanding of the basic principles of cryptography, and having studied how these principles are actually used, you are now in position to design your own secure network protocols!

Armed with the techniques covered in Sections 8.2 through 8.8, Bob and Alice can communicate securely. (One can only hope that they are networking students who have learned this material and can thus avoid having their tryst uncovered by Trudy!) But confidentiality is only a small part of the network security picture. As we learned in Section 8.9, increasingly, the focus in network security has been on securing the network infrastructure against a potential onslaught by the bad guys. In the latter part of this chapter, we thus covered firewalls and IDS systems which inspect packets entering and leaving an organization's network.

This chapter has covered a lot of ground, while focusing on the most important topics in modern network security. Readers who desire to dig deeper are encouraged to investigate the references cited in this chapter. In particular, we recommend [Skoudis 2006] for attacks and operational security, [Kaufman 1995] for cryptography and how it applies to network security, [Rescorla 2001] for an in-depth but readable treatment of SSL, and [Edney 2003] for a thorough discussion of 802.11 security, including an insightful investigation into WEP and its flaws.



Homework Problems and Questions

Chapter 8 Review Problems

SECTION 8.1

- R1. What are the differences between message confidentiality and message integrity? Can you have confidentiality without integrity? Can you have integrity without confidentiality? Justify your answer.
- R2. Internet entities (routers, switches, DNS servers, Web servers, user end systems, and so on) often need to communicate securely. Give three specific example pairs of Internet entities that may want secure communication.

SECTION 8.2

- R3. From a service perspective, what is an important difference between a symmetric-key system and a public-key system?
- R4. Suppose that an intruder has an encrypted message as well as the decrypted version of that message. Can the intruder mount a ciphertext-only attack, a known-plaintext attack, or a chosen-plaintext attack?
- R5. Consider an 8-block cipher. How many possible input blocks does this cipher have? How many possible mappings are there? If we view each mapping as a key, then how many possible keys does this cipher have?
- R6. Suppose n people want to communicate with each other. 1 other people using symmetric key encryption. All communication between any two people, i and j , is visible to all other people in this group and no other person in this group should be able to decode their communication. How many keys are required in the system as a whole? Now suppose that public key encryption is used. How many keys are required in this case?
- R7. Suppose $n = 10,000$, $a = 10,023$, and $b = 10,004$. Use an identity of modular arithmetic to calculate $a^{b \mod n}$.
- R8. Suppose you want to encrypt the message 10101111 by encrypting the decimal number that corresponds to the message. What is the decimal number?

SECTIONS 8.3...8.4

- R9. In what way does a hash provide a better message integrity check than a checksum (such as the Internet checksum)?
- R10. Can you decrypt a hash of a message to get the original message? Explain your answer.
- R11. Consider a variation of the MAC algorithm (Figure 8.9) where the sender sends $H(m) + s$, where $H(m) + s$ is the concatenation of $H(m)$ and s . Is this variation flawed? Why or why not?

- R12. What does it mean for a signed document to be verifiable and non-forgeable?
- R13. In what way does the public-key encrypted message hash provide a better digital signature than the public-key encrypted message?
- R14. Suppose certifier.com creates a certificate for foo.com. Typically, the entire certificate would be encrypted with certifier.com's public key. True or False?
- R15. Suppose Alice has a message that she is ready to send to anyone who asks. Thousands of people want to obtain Alice's message, but each wants to be sure of the integrity of the message. In this context, do you think a MAC-based or a digital-signature-based integrity scheme is more suitable? Why?
- R16. What is the purpose of a nonce in an end-point authentication protocol?
- R17. What does it mean to say that a nonce is a once-in-a-lifetime value? In whose lifetime?
- R18. Is the message integrity scheme based on HMAC susceptible to playback attacks? If so, how can a nonce be incorporated into the scheme to remove this susceptibility?

SECTIONS 8.5...8.8

- R19. Suppose that Bob receives a PGP message from Alice. How does Bob know for sure that Alice created the message (rather than, say, Trudy)? Does PGP use a MAC for message integrity?
- R20. In the SSL record, there is a field for SSL sequence numbers. True or False?
- R21. What is the purpose of the random nonces in the SSL handshake?
- R22. Suppose an SSL session employs a block cipher with CBC. True or False:
The server sends to the client the IV in the clear?
- R23. Suppose Bob initiates a TCP connection to Trudy who is pretending to be Alice. During the handshake, Trudy sends Bob Alice's certificate. In what step of the SSL handshake algorithm will Bob discover that he is not communicating with Alice?
- R24. Consider sending a stream of packets from Host A to Host B using IPsec. Typically, a new SA will be established for each packet sent in the stream. True or False?
- R25. Suppose that TCP is being run over IPsec between headquarters and the branch office in Figure 8.28. If TCP retransmits the same packet, then the two corresponding packets sent by R1 packets will have the same sequence number in the ESP header. True or False?
- R26. An IKE SA and an IPsec SA are the same thing. True or False?
- R27. Consider WEP for 802.11. Suppose that the data is 10101100 and the keystream is 11110000. What is the resulting ciphertext?
- R28. In WEP, an IV is sent in the clear in every frame. True or False?

SECTION 8.9

- R29. Stateful packet filters maintain two data structures. Name them and briefly describe what they do.
- R30. Consider a traditional (stateless) packet filter. This packet filter may filter packets based on TCP flag bits as well as other header fields. True or False?
- R31. In a traditional packet filter, each interface can have its own access control list. True or False?
- R32. Why must an application gateway work in conjunction with a router filter to be effective?
- R33. Signature-based IDSs and IPSs inspect into the payloads of TCP and UDP segments. True or False?



Problems

- P1. Using the monoalphabetic cipher in Figure 8.3, encode the message •This is an easy problem.Ž Decode the message •rmij•u uamu xyj.Ž
- P2. Show that Trudy•s known-plaintext attack, in which she knows the (ciphertext, plaintext) translation pairs for seven letters, reduces the number of possible substitutions to be checked in the example in Section 8.2.1 by approximately 10⁷.
- P3. Consider the polyalphabetic system shown in Figure 8.4. Will a chosen-plaintext attack that is able to get the plaintext encoding of the message •The quick brown fox jumps over the lazy dog.Ž be sufficient to decode all messages? Why or why not?
- P4. Consider the block cipher in Figure 8.5. Suppose that each block cipher simply reverses the order of the eight input bits (so that, for example, 11110000 becomes 000011). Further suppose that the 64-bit scrambler does not modify any bits (so that the output value of the n th bit is equal to the input value of the n th bit). (a) With $n = 3$ and the original 64-bit input equal to 10100000 repeated eight times, what is the value of the output? (b) Repeat part (a) but now change the last bit of the original 64-bit input from a 0 to a 1. (c) Repeat parts (a) and (b) but now suppose that the 64-bit scrambler inverses the order of the 64 bits.
- P5. Consider the block cipher in Figure 8.5. For a given •keyŽ Alice and Bob would need to keep eight tables, each 8 bits by 8 bits. For Alice (or Bob) to store all eight tables, how many bits of storage are necessary? How does this number compare with the number of bits required for a full-table 64-bit block cipher?
- P6. Consider the 3-bit block cipher in Table 8.1. Suppose the plaintext is 100100100. (a) Initially assume that CBC is not used. What is the resulting

ciphertext? (b) Suppose Trudy sniffs the ciphertext. Assuming she knows that a 3-bit block cipher without CBC is being employed (but doesn't know the specific cipher), what can she surmise? (c) Now suppose that CBC is used with IV = 111. What is the resulting ciphertext?

P7. (a) Using RSA, choose $p = 3$ and $q = 11$, and encode the word "dog" by encrypting each letter separately. Apply the decryption algorithm to the encrypted version to recover the original plaintext message. (b) Repeat part (a) but now encrypt "dog" as one message.

P8. Consider RSA with $p = 5$ and $q = 11$.

- What are e and n ?
- Let e be 3. Why is this an acceptable choice?
- Find d such that $de \equiv 1 \pmod{n}$ and $d < 160$.
- Encrypt the message "cat" using the key (n, e) . Let c denote the corresponding ciphertext. Show all work. Hint: To simplify the calculations, use the fact:

$$[(a \bmod n) \in k \bmod n] \bmod n = (a \in b) \bmod n$$

P9. In this problem, we explore the Diffie-Hellman (DH) public-key encryption algorithm, which allows two entities to agree on a shared key. The DH algorithm makes use of a large prime number p and another large number g less than p . Both p and g are made public (so that an attacker would know them). In DH, Alice and Bob each independently choose secret keys S_A and S_B , respectively. Alice then computes her public key by raising g to S_A and then taking mod p . Bob similarly computes his own public key by raising g to S_B and then taking mod p . Alice and Bob then exchange their public keys over the Internet. Alice then calculates the shared key by raising T_B to S_A and then taking mod p . Similarly, Bob calculates the shared key by raising T_A to S_B and then taking mod p .

- Prove that, in general, Alice and Bob obtain the same symmetric key, that is, prove $S = S'$.
- With $p = 11$ and $g = 2$, suppose Alice and Bob choose private keys $S_A = 5$ and $S_B = 12$, respectively. Calculate Alice's and Bob's public keys and T_A and T_B . Show all work.
- Following up on part (b), now calculate the shared symmetric key. Show all work.
- Provide a timing diagram that shows how Diffie-Hellman can be attacked by a man-in-the-middle. The timing diagram should have three vertical lines, one for Alice, one for Bob, and one for the attacker Trudy.

P10. Suppose Alice wants to communicate with Bob using symmetric key cryptography using a session key. In Section 8.2, we learned how public-key

cryptography can be used to distribute the session key from Alice to Bob. In this problem, we explore how the session key can be distributed, without public key cryptography, using a key distribution center (KDC). The KDC is a server that shares a unique secret symmetric key with each registered user. For Alice and Bob, denote these keys by K_{A-KDC} and K_{B-KDC} . Design a scheme that uses the KDC to distribute K to Alice and Bob. Your scheme should use three messages to distribute the session key: a message from Alice to the KDC; a message from the KDC to Alice; and finally a message from Alice to Bob. The first message is (A, B) . Using the notation K_{A-KDC} , K_{B-KDC} , S , A , and B answer the following questions.

- What is the second message?
- What is the third message?

- P11. Compute a third message, different from the two messages in Figure 8.8, that has the same checksum as the messages in Figure 8.8.
- P12. Suppose Alice and Bob share two secret keys: an authentication key and a symmetric encryption key. Augment Figure 8.9 so that both integrity and confidentiality are provided.
- P13. In the BitTorrent P2P file distribution protocol (see Chapter 2), the seed breaks the file into blocks, and the peers redistribute the blocks to each other. Without any protection, an attacker can easily wreak havoc in a torrent by masquerading as a benevolent peer and sending bogus blocks to a small subset of peers in the torrent. These unsuspecting peers then redistribute the bogus blocks to other peers, which in turn redistribute the bogus blocks to even more peers. Thus, it is critical for BitTorrent to have a mechanism that allows a peer to verify the integrity of a block, so that it doesn't redistribute bogus blocks. Assume that when a peer joins a torrent, it initially gets a .torrent file from a fully trusted source. Describe a simple scheme that allows peers to verify the integrity of blocks.
- P14. The OSPF routing protocol uses a MAC rather than digital signatures to provide message integrity. Why do you think a MAC was chosen over digital signatures?
- P15. Consider our authentication protocol in Figure 8.18 in which Alice authenticates herself to Bob, which we saw works well (i.e., we found no flaws in it). Now suppose that while Alice is authenticating herself to Bob, Bob must authenticate himself to Alice. Give a scenario by which Trudy, pretending to be Alice, can now authenticate herself to Bob as Alice. Consider that the sequence of operations of the protocol, one with Trudy initiating and one with Bob initiating, can be arbitrarily interleaved. Pay particular attention to the fact that both Bob and Alice will use a nonce, and that if care is not taken, the same nonce can be used maliciously.)
- P16. A natural question is whether we can use a nonce and public key cryptography to solve the end-point authentication problem in Section 8.4. Consider

the following natural protocol: (1) Alice sends the message "Alice" to Bob. (2) Bob chooses a nonce and sends it to Alice. (3) Alice uses her private key to encrypt the nonce and sends the resulting value to Bob. (4) Bob applies Alice's public key to the received message. Thus, Bob computes R and authenticates Alice.

- Diagram this protocol, using the notation for public and private keys employed in the textbook.
 - Suppose that certificates are not used. Describe how Trudy can become a "woman-in-the-middle" by intercepting Alice's messages and then pretending to be Alice to Bob.
- P17. Figure 8.19 shows the operations that Alice must perform with PGP to provide confidentiality, authentication, and integrity. Diagram the corresponding operations that Bob must perform on the package received from Alice.
- P18. Suppose Alice wants to send an e-mail to Bob. Bob has a public-private key pair (K_B^+, K_B^-), and Alice has Bob's certificate. But Alice does not have a public, private key pair. Alice and Bob (and the entire world) share the same hash function $H()$.
- In this situation, is it possible to design a scheme so that Bob can verify that Alice created the message? If so, show how with a block diagram for Alice and Bob.
 - Is it possible to design a scheme that provides confidentiality for sending the message from Alice to Bob? If so, show how with a block diagram for Alice and Bob.
- P19. Consider the Wireshark output below for a portion of an SSL session.
- Is Wireshark packet 112 sent by the client or server?
 - What is the server's IP address and port number?
 - Assuming no loss and no retransmissions, what will be the sequence number of the next TCP segment sent by the client?
 - How many SSL records does Wireshark packet 112 contain?
 - Does packet 112 contain a Master Secret or an Encrypted Master Secret or neither?
 - Assuming that the handshake type field is 1 byte and each length field is 3 bytes, what are the values of the first and last bytes of the Master Secret (or Encrypted Master Secret)?
 - The client encrypted handshake message takes into account how many SSL records?
 - The server encrypted handshake message takes into account how many SSL records?
- P20. In Section 8.6.1, it is shown that without sequence numbers, Trudy (a woman-in-the middle) can wreak havoc in an SSL session by interchanging TCP

(Wireshark screenshot reprinted by permission of the Wireshark Foundation.)

segments. Can Trudy do something similar by deleting a TCP segment? What does she need to do to succeed at the deletion attack? What effect will it have?

- P21. Suppose Alice and Bob are communicating over an SSL session. Suppose an attacker, who does not have any of the shared keys, inserts a bogus TCP segment into a packet stream with correct TCP checksum and sequence numbers (and correct IP addresses and port numbers). Will SSL at the receiving side accept the bogus packet and pass the payload to the receiving application? Why or why not?
- P22. The following True/False questions pertain to Figure 8.28.
- When a host in 172.16.1/24 sends a datagram to an Amazon.com server, the router R1 will encrypt the datagram using IPsec.
 - When a host in 172.16.1/24 sends a datagram to a host in 172.16.2/24, the router R1 will change the source and destination address of the IP datagram.
 - Suppose a host in 172.16.1/24 initiates a TCP connection to a Web server in 172.16.2/24. As part of this connection, all datagrams sent by R1 will have protocol number 50 in the left-most IPv4 header field.

- d. Consider sending a TCP segment from a host in 172.16.1/24 to a host in 172.16.2/24. Suppose the acknowledgment for this segment gets lost, so that TCP resends the segment. Because IPsec uses sequence numbers, R1 will not resend the TCP segment.
- P23. Consider the example in Figure 8.28. Suppose Trudy is a woman-in-the-middle, who can insert datagrams into the stream of datagrams going from R1 and R2. As part of a replay attack, Trudy sends a duplicate copy of one of the datagrams sent from R1 to R2. Will R2 decrypt the duplicate datagram and forward it into the branch-office network? If not, describe in detail how R2 detects the duplicate datagram.
- P24. Consider the following pseudo-WEP protocol. The key is 4 bits and the IV is 2 bits. The IV is appended to the end of the key when generating the keystream. Suppose that the shared secret key is 1010. The keystreams for the four possible inputs are as follows:
- 101000: 00101011010101001011010100100 . . .
- 101001: 1010011011001010110100100101101 . . .
- 101010: 00011010001100010100101001111 . . .
- 101011: 1111010101000000101010100010111 . . .
- Suppose all messages are 8-bits long. Suppose the ICV (integrity check) is 4-bits long, and is calculated by XOR-ing the first 4 bits of data with the last 4 bits of data. Suppose the pseudo-WEP packet consists of three fields: first the IV field, then the message field, and last the ICV field, with some of these fields encrypted.
- We want to send the message 10100000 using the IV = 11 and using WEP. What will be the values in the three WEP fields?
 - Show that when the receiver decrypts the WEP packet, it recovers the message and the ICV.
 - Suppose Trudy intercepts a WEP packet (not necessarily with the IV = 11) and wants to modify it before forwarding it to the receiver. Suppose Trudy flips the first ICV bit. Assuming that Trudy does not know the keystreams for any of the IVs, what other bit(s) must Trudy also flip so that the received packet passes the ICV check?
 - Justify your answer by modifying the bits in the WEP packet in part (a), decrypting the resulting packet, and verifying the integrity check.
- P25. Provide a filter table and a connection table for a stateful firewall that is as restrictive as possible but accomplishes the following:
- Allows all internal users to establish Telnet sessions with external hosts.
 - Allows external users to surf the company Web site at 222.22.0.12.
 - But otherwise blocks all inbound and outbound traffic.

The internal network is 222.22/16. In your solution, suppose that the connection table is currently caching three connections, all from inside to outside. You'll need to invent appropriate IP addresses and port numbers.

- P26. Suppose Alice wants to visit the Web site activist.com using a TOR-like service. This service uses two non-colluding proxy servers, Proxy1 and Proxy2. Alice first obtains the certificates (each containing a public key) for Proxy1 and Proxy2 from some central server. Denote $K_1^e()$, $K_1^d()$, and $K_2^e()$ for the encryption/decryption with public and private RSA keys.
- Using a timing diagram, provide a protocol (as simple as possible) that enables Alice to establish a shared session key with Proxy1. Denote $S_1(m)$ for encryption/decryption of data with the shared key $\$_1$.
 - Using a timing diagram, provide a protocol (as simple as possible) that allows Alice to establish a shared session key with Proxy2 without revealing her IP address to Proxy2.
 - Assume now that shared keys $\$_1$ and $\$_2$ are now established. Using a timing diagram, provide a protocol (as simple as possible) that allows Alice to request an html page from activist.com without revealing her IP address to Proxy2 and without revealing to Proxy1 which site she is visiting. Your diagram should end with an HTTP request arriving at activist.com.



Wireshark Lab

In this lab (available from the companion Web site), we investigate the Secure Sockets Layer (SSL) protocol. Recall from Section 8.6 that SSL is used for securing a TCP connection, and that it is extensively used in practice for secure Internet transactions. In this lab, we will focus on the SSL records sent over the TCP connection. We will attempt to delineate and classify each of the records, with a goal of understanding the why and how for each record. We investigate the various SSL record types as well as the fields in the SSL messages. We do so by analyzing a trace of the SSL records sent between your host and an e-commerce server.



IPsec Lab

In this lab (available from the companion Web site), we will explore how to create IPsec SAs between linux boxes. You can do the first part of the lab with two ordinary linux boxes, each with one Ethernet adapter. But for the second part of the lab, you will need four linux boxes, two of which having two Ethernet adapters. In the second half of the lab, you will create IPsec SAs using the ESP protocol in the tunnel mode. You will do this by first manually creating the SAs, and then by having IKE create the SAs.

AN INTERVIEW WITH . . .

Steven M. Bellovin

Steven M. Bellovin joined the faculty at Columbia University after many years at the Network Services Research Lab at AT&T Labs Research in Florham Park, New Jersey. His focus is on networks, security, and why the two are incompatible. In 1995, he was awarded the Usenix Lifetime Achievement Award for his work in the creation of Usenet, the first newsgroup exchange network that linked two or more computers and allowed users to share information and join in discussions. Steve is also an elected member of the National Academy of Engineering. He received his BA from Columbia University and his PhD from the University of North Carolina at Chapel Hill.



What led you to specialize in the networking security area?

This is going to sound odd, but the answer is simple: It was fun. My background was in systems programming and systems administration, which leads fairly naturally to security. And I've always been interested in communications, ranging back to part-time systems programming jobs when I was in college.

My work on security continues to be motivated by two things, a desire to keep computers useful, which means that their function can't be corrupted by attackers, and a desire to protect privacy.

What was your vision for Usenet at the time that you were developing it? And now?

We originally viewed it as a way to talk about computer science and computer programming around the country, with a lot of local use for administrative matters, for-sale ads, and so on. In fact, my original prediction was one to two messages per day, from 50...100 sites at the most, ever. But the real growth was in people-related topics, including, but not limited to, human interactions with computers. My favorite newsgroups, over the years, have been things like rec.woodworking, as well as sci.crypt.

To some extent, netnews has been displaced by the Web. Were I to start designing it today, it would look very different. But it still excels as a way to reach a very broad audience that is interested in the topic, without having to rely on particular Web sites.

Has anyone inspired you professionally? In what ways?

Professor Fred Brooks, the founder and original chair of the computer science department at the University of North Carolina at Chapel Hill, the manager of the team that developed the IBM S/360 and OS/360, and the author of *The Mythical Man-Month*, was a tremendous influence on my career. More than anything else, he taught outlook and trade-offs, how to

look at problems in the context of the real world (and how much messier the real world is than a theorist would like), and how to balance competing interests in designing a solution. Most computer work is engineering, the art of making the right trade-offs to satisfy many contradictory objectives.

What is your vision for the future of networking and security?

Thus far, much of the security we have has come from isolation. A firewall, for example, works by cutting off access to certain machines and services. But we're in an era of increasing connectivity, it's gotten harder to isolate things. Worse yet, our production systems require far more separate pieces, interconnected by networks. Securing all that is one of our biggest challenges.

What would you say have been the greatest advances in security? How much further do we have to go?

At least scientifically, we know how to do cryptography. That's been a big help. But most security problems are due to buggy code, and that's a much harder problem. In fact, it's the oldest unsolved problem in computer science, and I think it will remain that way. The challenge is figuring out how to secure systems when we have to build them out of insecure components. We can already do that for reliability in the face of hardware failures; can we do the same for security?

Do you have any advice for students about the Internet and networking security?

Learning the mechanisms is the easy part. Learning how to think paranoid is harder. You have to remember that probability distributions don't apply, the attackers can and will find improbable conditions. And the details matter, a lot.



Network Management

Having made our way through the first eight chapters of this text, we're now well aware that a network consists of *many* complex, interacting pieces of hardware and software—from the links, switches, routers, hosts, and other devices that comprise the physical components of the network to the many protocols (in both hardware and software) that control and coordinate these devices. When hundreds or thousands of such components are cobbled together by an organization to form a network, it is not surprising that components will occasionally malfunction, that network elements will be misconfigured, that network resources will be overutilized, or that network components will simply “break” (for example, a cable will be cut or a can of soda will be spilled on top of a router). The network administrator, whose job it is to keep the network “up and running,” must be able to respond to (and better yet, avoid) such mishaps. With potentially thousands of network components spread out over a wide area, the network administrator in a network operations center (NOC) clearly needs tools to help monitor, manage, and control the network. In this chapter, we'll examine the architecture, protocols, and information base used by a network administrator in this task.

9.1 What Is Network Management?

Before diving in to network management itself, let's first consider a few illustrative "real-world" non-networking scenarios in which a complex system with many interacting components must be monitored, managed, and controlled by an administrator. Electrical power-generation plants have a control room where dials, gauges, and lights monitor the status (temperature, pressure, flow) of remote valves, pipes, vessels, and other plant components. These devices allow the operator to monitor the plant's many components, and may alert the operator (with the famous flashing red warning light) when trouble is imminent. Actions are taken by the plant operator to control these components. Similarly, an airplane cockpit is instrumented to allow a pilot to monitor and control the many components that make up an airplane. In these two examples, the "administrator" *monitors* remote devices and *analyzes* their data to ensure that they are operational and operating within prescribed limits (for example, that a core meltdown of a nuclear power plant is not imminent, or that the plane is not about to run out of fuel), *reactively controls* the system by making adjustments in response to the changes within the system or its environment, and *proactively manages* the system (for example, by detecting trends or anomalous behavior, allowing action to be taken before serious problems arise). In a similar sense, the network administrator will actively monitor, manage, and control the system with which she or he is entrusted.

In the early days of networking, when computer networks were research artifacts rather than a critical infrastructure used by hundreds of millions of people a day, "network management" was unheard of. If one encountered a network problem, one might run a few pings to locate the source of the problem and then modify system settings, reboot hardware or software, or call a remote colleague to do so. (A very readable discussion of the first major "crash" of the ARPAnet on October 27, 1980, long before network management tools were available, and the efforts taken to recover from and understand the crash is [RFC 789].) As the public Internet and private intranets have grown from small networks into a large global infrastructure, the need to manage the huge number of hardware and software components within these networks more systematically has grown more important as well.

In order to motivate our study of network management, let's begin with a simple example. Figure 9.1 illustrates a small network consisting of three routers and a number of hosts and servers. Even in such a simple network, there are many scenarios in which a network administrator might benefit tremendously from having appropriate network management tools:

- *Detecting failure of an interface card at a host or a router.* With appropriate network management tools, a network entity (for example, router A) may report to the network administrator that one of its interfaces has gone down. (This is certainly preferable to a phone call to the NOC from an irate user who says the network connection is down!) A network administrator who actively monitors

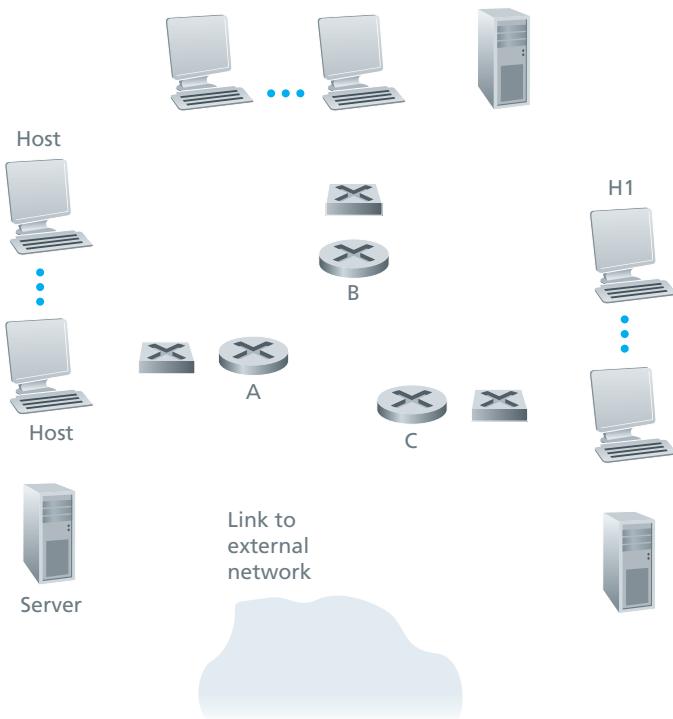


Figure 9.1 ♦ A simple scenario illustrating the uses of network management

and analyzes network traffic may be able to *really* impress the would-be irate user by detecting problems in the interface ahead of time and replacing the interface card before it fails. This might be done, for example, if the administrator noted an increase in checksum errors in frames being sent by the soon-to-die interface.

- *Host monitoring.* Here, the network administrator might periodically check to see if all network hosts are up and operational. Once again, the network administrator may really be able to impress a network user by proactively responding to a problem (host down) before it is reported by a user.
- *Monitoring traffic to aid in resource deployment.* A network administrator might monitor source-to-destination traffic patterns and notice, for example, that by switching servers between LAN segments, the amount of traffic that crosses multiple LANs could be significantly decreased. Imagine the happiness all around when better performance is achieved with no new equipment costs. Similarly, by monitoring link utilization, a network administrator might determine that a LAN segment or the external link to the outside world is overloaded and

that a higher-bandwidth link should thus be provisioned (alas, at an increased cost). The network administrator might also want to be notified automatically when congestion levels on a link exceed a given threshold value, in order to provision a higher-bandwidth link before congestion becomes serious.

- *Detecting rapid changes in routing tables.* Route flapping—frequent changes in the routing tables—may indicate instabilities in the routing or a misconfigured router. Certainly, the network administrator who has improperly configured a router would prefer to discover the error him- or herself, before the network goes down.
- *Monitoring for SLAs.* **Service Level Agreements (SLAs)** are contracts that define specific performance metrics and acceptable levels of network-provider performance with respect to these metrics [Huston 1999a]. Verizon and Sprint are just two of the many network providers that guarantee SLAs [AT&T SLA 2012; Verizon SLA 2012] to their customers. These SLAs include service availability (outage), latency, throughput, and outage notification requirements. Clearly, if performance criteria are to be part of a service agreement between a network provider and its users, then measuring and managing performance will be of great importance to the network administrator.
- *Intrusion detection.* A network administrator may want to be notified when network traffic arrives from, or is destined for, a suspicious source (for example, host or port number). Similarly, a network administrator may want to detect (and in many cases filter) the existence of certain types of traffic (for example, source-routed packets, or a large number of SYN packets directed to a given host) that are known to be characteristic of the types of security attacks that we considered in Chapter 8.

The International Organization for Standardization (ISO) has created a network management model that is useful for placing the anecdotal scenarios above in a more structured framework. Five areas of network management are defined:

- *Performance management.* The goal of performance management is to quantify, measure, report, analyze, and control the performance (for example, utilization and throughput) of different network components. These components include individual devices (for example, links, routers, and hosts) as well as end-to-end abstractions such as a path through the network. We will see shortly that protocol standards such as the Simple Network Management Protocol (SNMP) [RFC 3410] play a central role in Internet performance management.
- *Fault management.* The goal of fault management is to log, detect, and respond to fault conditions in the network. The line between fault management and performance management is rather blurred. We can think of fault management as the immediate handling of transient network failures (for example, link, host, or router hardware or software outages), while performance management takes

the longer-term view of providing acceptable levels of performance in the face of varying traffic demands and occasional network device failures. As with performance management, the SNMP protocol plays a central role in fault management.

- *Configuration management.* Configuration management allows a network manager to track which devices are on the managed network and the hardware and software configurations of these devices. An overview of configuration management and requirements for IP-based networks can be found in [RFC 3139].
- *Accounting management.* Accounting management allows the network manager to specify, log, and control user and device access to network resources. Usage quotas, usage-based charging, and the allocation of resource-access privileges all fall under accounting management.
- *Security management.* The goal of security management is to control access to network resources according to some well-defined policy. The key distribution centers that we studied in Section 8.3 are components of security management. The use of firewalls to monitor and control external access points to one's network, a topic we studied in Section 8.9, is another crucial component.

In this chapter, we'll cover only the rudiments of network management. Our focus will be purposefully narrow—we'll examine only the *infrastructure* for network management—the overall architecture, network management protocols, and information base through which a network administrator keeps the network up and running. We'll *not* cover the decision-making processes of the network administrator, who must plan, analyze, and respond to the management information that is conveyed to the NOC. In this area, topics such as fault identification and management [Katzela 1995; Medhi 1997; Labovitz 1997; Steinder 2002; Feamster 2005; Wu 2005; Teixeira 2006], anomaly detection [Lakhina 2004; Lakhina 2005; Barford 2009], and more come into consideration. Nor will we cover the broader topic of service management [Saydam 1996; RFC 3052]—the provisioning of resources such as bandwidth, server capacity, and the other computational/communication resources needed to meet the mission-specific service requirements of an enterprise.

An often-asked question is “What is network management?” Our discussion above has motivated the need for, and illustrated a few of the uses of, network management. We'll conclude this section with a single-sentence (albeit a rather long run-on sentence) definition of network management from [Saydam 1996]:

“Network management includes the deployment, integration, and coordination of the hardware, software, and human elements to monitor, test, poll, configure, analyze, evaluate, and control the network and element resources to meet the real-time, operational performance, and Quality of Service requirements at a reasonable cost.”

It's a mouthful, but it's a good workable definition. In the following sections, we'll add some meat to this rather bare-bones definition of network management.

9.2 The Infrastructure for Network Management

We've seen in the preceding section that network management requires the ability to "monitor, test, poll, configure, . . . and control" the hardware and software components in a network. Because the network devices are distributed, this will, at a minimum, require that the network administrator be able to gather data (for example, for monitoring purposes) from a remote entity and effect changes at that remote entity (for example, control it). A human analogy will prove useful here for understanding the infrastructure needed for network management.

Imagine that you're the head of a large organization that has branch offices around the world. It's your job to make sure that the pieces of your organization are operating smoothly. How will you do so? At a minimum, you'll periodically gather data from your branch offices in the form of reports and various quantitative measures of activity, productivity, and budget. You'll occasionally (but not always) be explicitly notified when there's a problem in one of the branch offices; the branch manager who wants to climb the corporate ladder (perhaps to get your job) may send you unsolicited reports indicating how smoothly things are running at his or her branch. You'll sift through the reports you receive, hoping to find smooth operations everywhere but no doubt finding problems in need of your attention. You might initiate a one-on-one dialogue with one of your problem branch offices, gather more data in order to understand the problem, and then pass down an executive order ("Make this change!") to the branch office manager.

Implicit in this very common human scenario is an infrastructure for controlling the organization—the boss (you), the remote sites being controlled (the branch offices), your remote agents (the branch office managers), communication protocols (for transmitting standard reports and data, and for one-on-one dialogues), and data (the report contents and the quantitative measures of activity, productivity, and budget). Each of these components in human organizational management has a counterpart in network management.

The architecture of a network management system is conceptually identical to this simple human organizational analogy. The network management field has its own specific terminology for the various components of a network management architecture, and so we adopt that terminology here. As shown in Figure 9.2, there are three principal components of a network management architecture: a managing entity (the boss in our analogy above—you), the managed devices (the branch office), and a network management protocol.

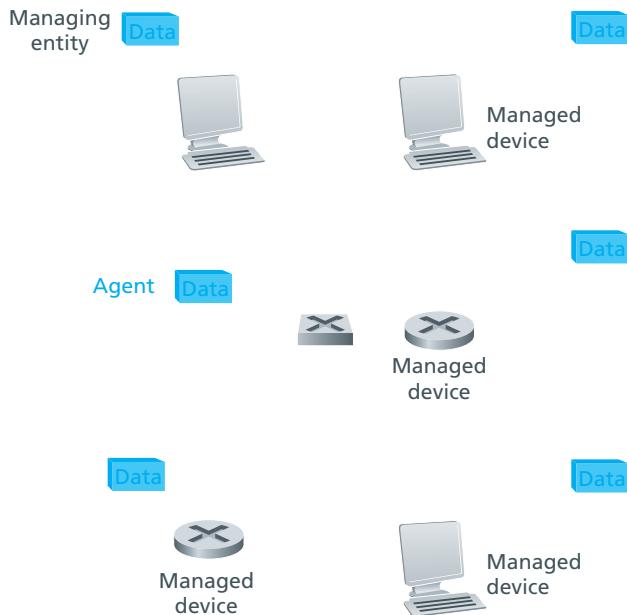


Figure 9.2 ♦ Principal components of a network management architecture

The **managing entity** is an application, typically with a human in the loop, running in a centralized network management station in the NOC. The managing entity is the locus of activity for network management; it controls the collection, processing, analysis, and/or display of network management information. It is here that actions are initiated to control network behavior and here that the human network administrator interacts with the network devices.

A **managed device** is a piece of network equipment (including its software) that resides on a managed network. This is the branch office in our human analogy. A managed device might be a host, router, bridge, hub, printer, or modem. Within a managed device, there may be several so-called **managed objects**. These managed objects are the actual pieces of hardware within the managed device (for example, a network interface card), and the sets of configuration parameters for the pieces of hardware and software (for example, an intradomain routing protocol such as RIP). In our human analogy, the managed objects might be the departments within the branch office. These managed objects have pieces of information associated with them that are collected into a **Management Information Base**.

(**MIB**); we'll see that the values of these pieces of information are available to (and in many cases able to be set by) the managing entity. In our human analogy, the MIB corresponds to quantitative data (measures of activity, productivity, and budget, with the latter being settable by the managing entity!) exchanged between the branch office and the main office. We'll study MIBs in detail in Section 9.3. Finally, also resident in each managed device is a **network management agent**, a process running in the managed device that communicates with the managing entity, taking local actions at the managed device under the command and control of the managing entity. The network management agent is the branch manager in our human analogy.

The third piece of a network management architecture is the **network management protocol**. The protocol runs between the managing entity and the managed devices, allowing the managing entity to query the status of managed devices and indirectly take actions at these devices via its agents. Agents can use the network management protocol to inform the managing entity of exceptional events (for example, component failures or violation of performance thresholds). It's important to note that the network management protocol does not itself manage the network. Instead, it provides capabilities that a network administrator can use to manage ("monitor, test, poll, configure, analyze, evaluate, and control") the network. This is a subtle, but important, distinction.

Although the infrastructure for network management is conceptually simple, one can often get bogged down with the network-management-speak vocabulary of "managing entity," "managed device," "managing agent," and "Management Information Base." For example, in network-management-speak, in our simple host-monitoring scenario, "managing agents" located at "managed devices" are periodically queried by the "managing entity"—a simple idea, but a linguistic mouthful! With any luck, keeping in mind the human organizational analogy and its obvious parallels with network management will be of help as we continue through this chapter.

Our discussion of network management architecture above has been generic, and broadly applies to a number of the network management standards and efforts that have been proposed over the years. Network management standards began maturing in the late 1980s, with OSI **CMISE/CMIP** (the **Common Management Information Services Element/Common Management Information Protocol**) [Piscatello 1993; Stallings 1993; Glitho 1998] and the Internet **SNMP (Simple Network Management Protocol)** [RFC 3410; Stallings 1999; Rose 1996] emerging as the two most important standards [Subramanian 2000]. Both are designed to be independent of vendor-specific products or networks. Because SNMP was quickly designed and deployed at a time when the need for network management was becoming painfully clear, SNMP found widespread use and acceptance. Today, SNMP has emerged as the most widely used and deployed network management framework. We'll cover SNMP in detail in the following section.



PRINCIPLES IN PRACTICE

COMCAST'S NETWORK OPERATIONS CENTER

Comcast's world-class fiber-based IP network delivers converged products and services to 49 million combined video, data and voice customers. Comcast's network includes more than 618,000 plant route miles, 138,000 fiber route miles, 30,000 backbone miles, 122,000 optical nodes, and massive storage for the Comcast Content Delivery Network, which delivers a Video on Demand product of more than 134 Terabytes. Each part of Comcast's network, up to and including the customers' homes or businesses, is monitored by one of the company's Operations Centers.

Comcast operates two National Network Operations Centers that manage the national backbone, regional area networks, national applications and specific platforms supporting voice, data and video infrastructure for residential, commercial and wholesale customers. In addition, Comcast has three Divisional Operations Centers that manage the local infrastructure that supports all of their customers. Both the National and Divisional Operations Centers are accountable for proactively monitoring all aspects of their network and product performance on a 7 x 24 x 365 basis, utilizing common processes and systems. For example, various network events at the national and local levels have common pre-defined severity levels, recovery processes, and expected Mean Time to Restore objectives. The national and divisional centers can back up each other if a local issue impacts a site's operation. In addition, the National and Divisional Operations Centers have an extensive Virtual Private Network that allows engineers to securely access the network to remotely perform proactive or reactive network management activities.

Comcast's approach to network management involves five key areas: Performance Management, Fault Management, Configuration Management, Accounting Management and Security Management. **Performance Management** is focused on understanding



These screens show tools supporting correlation, threshold management, ticketing used by Comcast technicians (Courtesy of Comcast.)

(continues)

how the network/systems and applications (collectively referred to as the ecosystem) are performing with respect to pre-defined measures specific to time of day, day of week, or special events (e.g., storm surges or pay events, such as a boxing match). These pre-defined performance measures exist throughout the service path, from the customer's residence or business through the entire network, as well as the interface points to partners and peers. In addition, synthetic transactions are run to ensure the health of the ecosystem on a continual basis. **Fault Management** is defined as the ability to detect, log and understand anomalies that may impact customers. Comcast utilizes correlation engines to properly determine an event's severity and act appropriately, eliminating or remediating potential issues before they affect customers. **Configuration Management** makes sure appropriate versions of hardware and software are in place across all elements of the ecosystem. Keeping these elements at their peak "golden" levels helps them avoid unintended consequences. **Accounting Management** ensures that the operations centers have a clear understanding of the provisioning and utilization of the ecosystem. This is especially important to ensure that at all times the operations centers have the ability to re-route traffic effectively. **Security Management** ensures that the proper controls exist to ensure the ecosystem is effectively protected against inappropriate access.

Network Operations Centers and the ecosystem they support are not static. Engineering and Operations personnel are constantly re-evaluating the pre-defined performance measures and tools to ensure that the customers' expectations for operational excellence are met.

9.3 The Internet-Standard Management Framework

Contrary to what the name SNMP (Simple Network Management Protocol) might suggest, network management in the Internet is much more than just a protocol for moving management data between a management entity and its agents, and has grown to be much more complex than the word "simple" might suggest. The current Internet-Standard Management Framework traces its roots back to the Simple Gateway Monitoring Protocol, SGMP [RFC 1028]. SGMP was designed by a group of university network researchers, users, and managers, whose experience with SGMP allowed them to design, implement, and deploy SNMP in just a few months [Lynch 1993]—a far cry from today's rather drawn-out standardization process. Since then, SNMP has evolved from SNMPv1 through SNMPv2 to the most recent version, SNMPv3 [RFC 3410], released in April 1999 and updated in December 2002.

When describing any framework for network management, certain questions must inevitably be addressed:

- What (from a semantic viewpoint) is being monitored? And what form of control can be exercised by the network administrator?
- What is the specific form of the information that will be reported and/or exchanged?
- What is the communication protocol for exchanging this information?

Recall our human organizational analogy from the previous section. The boss and the branch managers will need to agree on the measures of activity, productivity, and budget used to report the branch office's status. Similarly, they'll need to agree on the actions the boss can take (for example, cut the budget, order the branch manager to change some aspect of the office's operation, or fire the staff and shut down the branch office). At a lower level of detail, they'll need to agree on the form in which this data is reported. For example, in what currency (dollars, euros?) will the budget be reported? In what units will productivity be measured? While these may seem like trivial details, they must be agreed upon, nonetheless. Finally, the manner in which information is conveyed between the main office and the branch offices (that is, their communication protocol) must be specified.

The Internet-Standard Management Framework addresses the questions posed above. The framework consists of four parts:

- Definitions of *network management objects*, known as MIB objects. In the Internet-Standard Management Framework, management information is represented as a collection of managed objects that together form a virtual information store, known as the Management Information Base (MIB). An MIB object might be a counter, such as the number of IP datagrams discarded at a router due to errors in an IP datagram header, or the number of carrier sense errors in an Ethernet interface card; descriptive information such as the version of the software running on a DNS server; status information such as whether a particular device is functioning correctly; or protocol-specific information such as a routing path to a destination. MIB objects thus define the management information maintained by a managed device. Related MIB objects are gathered into **MIB modules**. In our human organizational analogy, the MIB defines the information conveyed between the branch office and the main office.
- A *data definition language*, known as SMI (Structure of Management Information). SMI defines the data types, an object model, and rules for writing and revising management information. MIB objects are specified in this data definition language. In our human organizational analogy, the SMI is used to define the details of the *format* of the information to be exchanged.
- A *protocol*, **SNMP**. SNMP is used for conveying information and commands between a managing entity and an agent executing on behalf of that entity within a managed network device.
- *Security and administration capabilities*. The addition of these capabilities represents the major enhancement in SNMPv3 over SNMPv2.

The Internet network management architecture is thus modular by design, with a protocol-independent data definition language and MIB, and an MIB-independent protocol. Interestingly, this modular architecture was first put in place to ease the transition from an SNMP-based network management to a network management framework being developed by ISO, the competing network management architecture when SNMP was first conceived—a transition that never occurred. Over time, however, SNMP’s design modularity has allowed it to evolve through three major revisions, with each of the four major parts of SNMP discussed above evolving independently. Clearly, the right decision about modularity was made, even if for the wrong reason!

In the following subsections, we cover the four major components of the Internet-Standard Management Framework in more detail.

9.3.1 Structure of Management Information: SMI

The **Structure of Management Information, SMI** (a rather oddly named component of the network management framework whose name gives no hint of its functionality), is the language used to define the management information residing in a managed-network entity. Such a definition language is needed to ensure that the syntax and semantics of the network management data are well defined and unambiguous. Note that the SMI does not define a specific instance of the data in a managed-network entity, but rather the language in which such information is specified. The documents describing the SMI for SNMPv3 (which rather confusingly, is called SMIv2) are [RFC 2578; RFC 2579; RFC 2580]. Let’s examine the SMI in a bottom-up manner, starting with the base data types in the SMI. We’ll then look at how managed objects are described in SMI, then how related managed objects are grouped into modules.

SMI Base Data Types

RFC 2578 specifies the basic data types in the SMI MIB module-definition language. Although the SMI is based on the ASN.1 (Abstract Syntax Notation One) [ISO X.680 2002] object-definition language (see Section 9.4), enough SMI-specific data types have been added that SMI should be considered a data definition language in its own right. The 11 basic data types defined in RFC 2578 are shown in Table 9.1. In addition to these scalar objects, it is also possible to impose a tabular structure on an ordered collection of MIB objects using the SEQUENCE OF construct; see RFC 2578 for details. Most of the data types in Table 9.1 will be familiar (or self-explanatory) to most readers. The one data type we will discuss in more detail shortly is the OBJECT IDENTIFIER data type, which is used to name an object.

SMI Higher-Level Constructs

In addition to the basic data types, the SMI data definition language also provides higher-level language constructs.

Data Type	Description
INTEGER	32-bit integer, as defined in ASN.1, with a value between -2^{31} and $2^{31} - 1$ inclusive, or a value from a list of possible named constant values.
Integer32	32-bit integer with a value between -2^{31} and $2^{31} - 1$ inclusive.
Unsigned32	Unsigned 32-bit integer in the range 0 to $2^{32} - 1$ inclusive.
OCTET STRING	ASN.1-format byte string representing arbitrary binary or textual data, up to 65,535 bytes long.
OBJECT IDENTIFIER	ASN.1-format administratively assigned (structured name); see Section 9.3.2.
IPAddress	32-bit Internet address, in network-byte order.
Counter32	32-bit counter that increases from 0 to $2^{32} - 1$ and then wraps around to 0.
Counter64	64-bit counter.
Gauge32	32-bit integer that will not count above $2^{32} - 1$ nor decrease beyond 0 when increased or decreased.
TimeTicks	Time, measured in 1/100ths of a second since some event.
Opaque	Uninterpreted ASN.1 string, needed for backward compatibility.

Table 9.1 ♦ Basic data types of the SMI

The OBJECT-TYPE construct is used to specify the data type, status, and semantics of a managed object. Collectively, these managed objects contain the management data that lies at the heart of network management. There are more than 10,000 defined objects in various Internet RFCs [RFC 3410]. The OBJECT-TYPE construct has four clauses. The SYNTAX clause of an OBJECT-TYPE definition specifies the basic data type associated with the object. The MAX-ACCESS clause specifies whether the managed object can be read, be written, be created, or have its value included in a notification. The STATUS clause indicates whether the object definition is current and valid, obsolete (in which case it should not be implemented, as its definition is included for historical purposes only), or deprecated (obsolete, but implementable for interoperability with older implementations). The DESCRIPTION clause contains a human-readable textual definition of the object; this “documents” the purpose of the managed object and should provide all the semantic information needed to implement the managed object.

As an example of the OBJECT-TYPE construct, consider the `ipSystemStatsInDelivers` object-type definition from [RFC 4293]. This object defines a 32-bit counter that keeps track of the number of IP datagrams that were received at the managed device and were successfully delivered to an upper-layer protocol.

The final line of this definition is concerned with the name of this object, a topic we'll consider in the following subsection.

```
ipSystemStatsInDelivers OBJECT-TYPE
    SYNTAX      Counter32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The total number of datagrams successfully
         delivered to IPuser-protocols (including ICMP)."

When tracking interface statistics, the counter
of the interface to which these datagrams were
addressed is incremented. This interface might
not be the same as the input interface for
some of the datagrams.

Discontinuities in the value of this counter can
occur at re-initialization of the management
system, and at other times as indicated by the
value of ipSystemStatsDiscontinuityTime."
::= { ipSystemStatsEntry 18 }
```

The MODULE-IDENTITY construct allows related objects to be grouped together within a “module.” For example, [RFC 4293] specifies the MIB module that defines managed objects (including `ipSystemStatsInDelivers`) for managing implementations of the Internet Protocol (IP) and its associated Internet Control Message Protocol (ICMP). [RFC 4022] specifies the MIB module for TCP, and [RFC 4113] specifies the MIB module for UDP. [RFC 4502] defines the MIB module for RMON remote monitoring. In addition to containing the OBJECT-TYPE definitions of the managed objects within the module, the MODULE-IDENTITY construct contains clauses to document contact information of the author of the module, the date of the last update, a revision history, and a textual description of the module. As an example, consider the module definition for management of the IP protocol:

```
ipMIB MODULE-IDENTITY
LAST-UPDATED "200602020000Z"
ORGANIZATION "IETF IPv6 MIB Revision Team"
CONTACT-INFO
    "Editor:
     Shawn A. Routhier
     Interworking Labs
     108 Whispering Pines Dr. Suite 235"
```

Scotts Valley, CA 95066
USA
EMail: <sar@iwl.com>"

DESCRIPTION

"The MIB module for managing IP and ICMP implementations, but excluding their management of IP routes.

Copyright (C) The Internet Society (2006).
This version of this MIB module is part of
RFC 4293; see the RFC itself for full legal
notices."

REVISION "200602020000Z"

DESCRIPTION

"The IP version neutral revision with added IPv6 objects for ND, default routers, and router advertisements. As well as being the successor to RFC 2011, this MIB is also the successor to RFCs 2465 and 2466. Published as RFC 4293."

REVISION "199411010000Z"

DESCRIPTION

"A separate MIB module (IP-MIB) for IP and ICMP management objects. Published as RFC 2011."

REVISION "199103310000Z"

DESCRIPTION

"The initial revision of this MIB module was part of MIB-II, which was published as RFC 1213."

::= { mib-2 48}

The NOTIFICATION-TYPE construct is used to specify information regarding SNMPv2-Trap and InformationRequest messages generated by an agent, or a managing entity; see Section 9.3.3. This information includes a textual DESCRIPTION of when such messages are to be sent, as well as a list of values to be included in the message generated; see [RFC 2578] for details. The MODULE-COMPLIANCE construct defines the set of managed objects within a module that an agent must implement. The AGENT-CAPABILITIES construct specifies the capabilities of agents with respect to object- and event-notification definitions.

9.3.2 Management Information Base: MIB

As noted previously, the **Management Information Base, MIB**, can be thought of as a virtual information store, holding managed objects whose values collectively reflect the current “state” of the network. These values may be queried and/or set by a managing entity by sending SNMP messages to the agent that is executing in a managed device on behalf of the managing entity. Managed objects are specified using the OBJECT-TYPE SMI construct discussed above and gathered into **MIB modules** using the MODULE-IDENTITY construct.

The IETF has been busy standardizing the MIB modules associated with routers, hosts, and other network equipment. This includes basic identification data about a particular piece of hardware, and management information about the device’s network interfaces and protocols. As of 2006 there were more than 200 standards-based MIB modules and an even larger number of vendor-specific (private) MIB modules. With all of these standards, the IETF needed a way to identify and name the standardized modules as well as the specific managed objects within a module. Rather than start from scratch, the IETF adopted a standardized object identification (naming) framework that had already been put in place by the International Organization for Standardization (ISO). As is the case with many standards bodies, the ISO had “grand plans” for its standardized object identification framework—to identify every possible standardized object (for example, data format, protocol, or piece of information) in any network, regardless of the network standards organization (for example, Internet IETF, ISO, IEEE, or ANSI), equipment manufacturer, or network owner. A lofty goal indeed! The object identification framework adopted by ISO is part of the ASN.1 (Abstract Syntax Notation One) [ISO X.680 2002] object definition language that we’ll discuss in Section 9.4. Standardized MIB modules have their own cozy corner in this all-encompassing naming framework, as discussed below.

As shown in Figure 9.3, objects are named in the ISO naming framework in a hierarchical manner. Note that each branch point in the tree has both a name and a number (shown in parentheses); any point in the tree is thus identifiable by the sequence of names or numbers that specify the path from the root to that point in the identifier tree. A fun, but incomplete and unofficial, Web-based utility for traversing part of the object identifier tree (using branch information contributed by volunteers) may be found in [OID Repository 2012].

At the top of the hierarchy are the ISO and the Telecommunication Standardization Sector of the International Telecommunication Union (ITU-T), the two main standards organizations dealing with ASN.1, as well as a branch for joint efforts by these two organizations. Under the ISO branch of the tree, we find entries for all ISO standards (1.0) and for standards issued by standards bodies of various ISO-member countries (1.2). Although not shown in Figure 9.3, under (ISO member body, a.k.a. 1.2) we would find USA (1.2.840), under which we would find a number of IEEE, ANSI, and company-specific standards. These include RSA (1.2.840.11359) and Microsoft (1.2.840.113556), under which we find the Microsoft File Formats (1.2.840.113556.4) for various Microsoft products, such as

Figure 9.3 ♦ ASN.1 object identifier tree

Word (1.2.840.113556.4.2). But we are interested here in networking (*not* Microsoft Word files), so let us turn our attention to the branch labeled 1.3, the standards issued by bodies recognized by the ISO. These include the U.S. Department of Defense (6) (under which we will find the Internet standards), the Open Software Foundation (22), the airline association SITA (69), NATO-identified bodies (57), as well as many other organizations.

Under the **Internet** branch of the tree (1.3.6.1), there are seven categories. Under the **private** (1.3.6.1.4) branch, we find a list [IANA 2009b] of the names and private enterprise codes of many thousands of private companies that have registered with the Internet Assigned Numbers Authority (IANA) [IANA 2009a]. Under the **management** (1.3.6.1.2) and **MIB-2** branches (1.3.6.1.2.1) of the object identifier tree, we find the definitions of the standardized MIB modules. Whew—it's a long journey down to our corner of the ISO name space!

Standardized MIB Modules

The lowest level of the tree in Figure 9.3 shows some of the important hardware-oriented MIB modules (**system** and **interface**) as well as modules associated

with some of the most important Internet protocols. [RFC 5000] lists all of the standardized MIB modules as of 2008. While MIB-related RFCs make for rather tedious and dry reading, it is instructive (that is, like eating vegetables, it is “good for you”) to consider a few MIB module definitions to get a flavor for the type of information in a module.

The managed objects falling under system contain general information about the device being managed; all managed devices must support the system MIB objects. Table 9.2 defines the objects in the system group, as defined in [RFC 1213]. Table 9.3 defines the managed objects in the MIB module for the UDP protocol at a managed entity.

9.3.3 SNMP Protocol Operations and Transport Mappings

The Simple Network Management Protocol version 2 (SNMPv2) [RFC 3416] is used to convey MIB information among managing entities and agents executing on behalf of managing entities. The most common usage of SNMP is in a **request-response mode** in which an SNMPv2 managing entity sends a request to an SNMPv2 agent, who receives the request, performs some action, and sends a reply to the request. Typically, a request will be used to query (retrieve) or modify (set) MIB object values

Object Identifier	Name	Type	Description (from RFC 1213)
1.3.6.1.2.1.1.1	<code>sysDescr</code>	OCTET STRING	“Full name and version identification of the system’s hardware type, software operating-system, and networking software.”
1.3.6.1.2.1.1.2	<code>sysObjectID</code>	OBJECT IDENTIFIER	Vendor-assigned object ID that “provides an easy and unambiguous means for determining ‘what kind of box’ is being managed.”
1.3.6.1.2.1.1.3	<code>sysUpTime</code>	TimeTicks	“The time (in hundredths of a second) since the network management portion of the system was last re-initialized.”
1.3.6.1.2.1.1.4	<code>sysContact</code>	OCTET STRING	“The contact person for this managed node, together with information on how to contact this person.”
1.3.6.1.2.1.1.5	<code>sysName</code>	OCTET STRING	“An administratively assigned name for this managed node. By convention, this is the node’s fully qualified domain name.”
1.3.6.1.2.1.1.6	<code>sysLocation</code>	OCTET STRING	“The physical location of this node.”
1.3.6.1.2.1.1.7	<code>sysServices</code>	Integer32	A coded value that indicates the set of services available at this node: physical (for example, a repeater), data link/subnet (for example, bridge), Internet (for example, IP gateway), end-to-end (for example, host), applications.

Table 9.2 ♦ Managed objects in the MIB-2 system group

Object Identifier	Name	Type	Description (from RFC 4113)
1.3.6.1.2.1.7.1	udpInDatagrams	Counter32	"total number of UDP datagrams delivered to UDP users"
1.3.6.1.2.1.7.2	udpNoPorts	Counter32	"total number of received UDP datagrams for which there was no application at the destination port"
1.3.6.1.2.1.7.3	udpInErrors	Counter32	"number of received UDP datagrams that could not be delivered for reasons other than the lack of an application at the destination port"
1.3.6.1.2.1.7.4	udpOutDatagrams	Counter32	"total number of UDP datagrams sent from this entity"

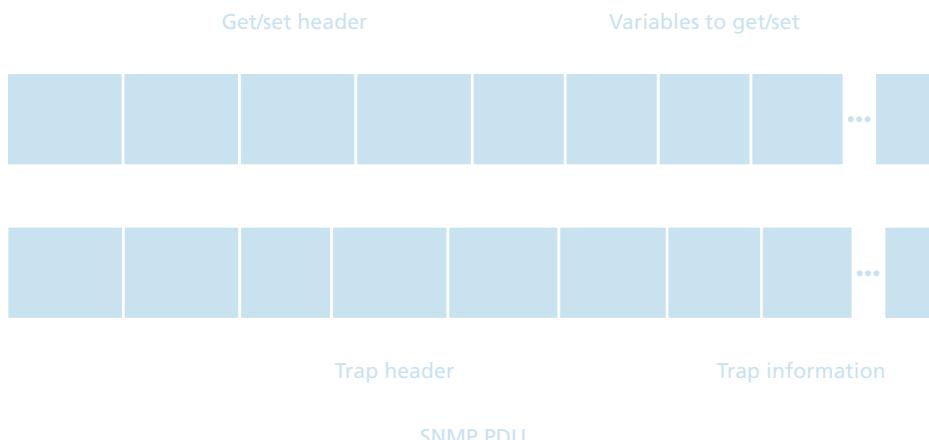
Table 9.3 ♦ Selected managed objects in the MIB-2 UDP module

associated with a managed device. A second common usage of SNMP is for an agent to send an unsolicited message, known as a **trap message**, to a managing entity. Trap messages are used to notify a managing entity of an exceptional situation that has resulted in changes to MIB object values. We saw earlier in Section 9.1 that the network administrator might want to receive a trap message, for example, when an interface goes down, congestion reaches a predefined level on a link, or some other noteworthy event occurs. Note that there are a number of important trade-offs between polling (request-response interaction) and trapping; see the homework problems.

SNMPv2 defines seven types of messages, known generically as protocol data units—PDUs—as shown in Table 9.4 and described next. The format of the PDU is shown in Figure 9.4.

- The **GetRequest**, **GetNextRequest**, and **GetBulkRequest** PDUs are all sent from a managing entity to an agent to request the value of one or more MIB objects at the agent’s managed device. The object identifiers of the MIB objects whose values are being requested are specified in the variable binding portion of the PDU. **GetRequest**, **GetNextRequest**, and **GetBulkRequest** differ in the granularity of their data requests. **GetRequest** can request an arbitrary set of MIB values; multiple **GetNextRequests** can be used to sequence through a list or table of MIB objects; **GetBulkRequest** allows a large block of data to be returned, avoiding the overhead incurred if multiple **GetRequest** or **GetNextRequest** messages were to be sent. In all three cases, the agent responds with a **Response** PDU containing the object identifiers and their associated values.
- The **SetRequest** PDU is used by a managing entity to set the value of one or more MIB objects in a managed device. An agent replies with a **Response** PDU with the “noError” error status to confirm that the value has indeed been set.

SNMPv2 PDU Type	Sender-receiver	Description
GetRequest	manager-to-agent	get value of one or more MIB object instances
GetNextRequest	manager-to-agent	get value of next MIB object instance in list or table
GetBulkRequest	manager-to-agent	get values in large block of data, for example, values in a large table
InformRequest	manager-to-manager	inform remote managing entity of MIB values remote to its access
SetRequest	manager-to-agent	set value of one or more MIB object instances
Response	agent-to-manager or manager-to-manager	generated in response to <code>GetRequest,</code> <code>GetNextRequest,</code> <code>GetBulkRequest,</code> <code>SetRequest</code> PDU, or <code>InformRequest</code>
SNMPv2-Trap	agent-to-manager	inform manager of an exceptional event

Table 9.4 ♦ SNMPv2 PDU types**Figure 9.4** ♦ SNMP PDU format

- The `InformRequest` PDU is used by a managing entity to notify another managing entity of MIB information that is remote to the receiving entity. The receiving entity replies with a `Response` PDU with the “noError” error status to acknowledge receipt of the `InformRequest` PDU.
- The final type of SNMPv2 PDU is the trap message. Trap messages are generated asynchronously; that is, they are *not* generated in response to a received request but rather in response to an event for which the managing entity requires notification. RFC 3418 defines well-known trap types that include a cold or warm start by a device, a link going up or down, the loss of a neighbor, or an authentication failure event. A received trap request has no required response from a managing entity.

Given the request-response nature of SNMPv2, it is worth noting here that although SNMP PDUs can be carried via many different transport protocols, the SNMP PDU is typically carried in the payload of a UDP datagram. Indeed, RFC 3417 states that UDP is “the preferred transport mapping.” Since UDP is an unreliable transport protocol, there is no guarantee that a request, or its response, will be received at the intended destination. The request ID field of the PDU is used by the managing entity to number its requests to an agent; an agent’s response takes its request ID from that of the received request. Thus, the request ID field can be used by the managing entity to detect lost requests or replies. It is up to the managing entity to decide whether to retransmit a request if no corresponding response is received after a given amount of time. In particular, the SNMP standard does not mandate any particular procedure for retransmission, or even if retransmission is to be done in the first place. It only requires that the managing entity “needs to act responsibly in respect to the frequency and duration of retransmissions.” This, of course, leads one to wonder how a “responsible” protocol should act!

9.3.4 Security and Administration

The designers of SNMPv3 have said that “SNMPv3 can be thought of as SNMPv2 with additional security and administration capabilities” [RFC 3410]. Certainly, there are changes in SNMPv3 over SNMPv2, but nowhere are those changes more evident than in the area of administration and security. The central role of security in SNMPv3 was particularly important, since the lack of adequate security resulted in SNMP being used primarily for monitoring rather than control (for example, `SetRequest` is rarely used in SNMPv1).

As SNMP has matured through three versions, its functionality has grown but so too, alas, has the number of SNMP-related standards documents. This is evidenced by the fact that there is even now an RFC [RFC 3411] that “describes an architecture for describing SNMP Management Frameworks”! While the notion of an “architecture” for “describing a framework” might be a bit much to wrap one’s mind around, the goal of RFC 3411 is an admirable one—to introduce a common language for describing the functionality and actions taken by an SNMPv3 agent or

managing entity. The architecture of an SNMPv3 entity is straightforward, and a tour through the architecture will serve to solidify our understanding of SNMP.

So-called **SNMP applications** consist of a command generator, notification receiver, and proxy forwarder (all of which are typically found in a managing entity); a command responder and notification originator (both of which are typically found in an agent); and the possibility of other applications. The command generator generates the `GetRequest`, `GetNextRequest`, `GetBulkRequest`, and `SetRequest` PDUs that we examined in Section 9.3.3 and handles the received responses to these PDUs. The command responder executes in an agent and receives, processes, and replies (using the `Response` message) to received `GetRequest`, `GetNextRequest`, `GetBulkRequest`, and `SetRequest` PDUs. The notification originator application in an agent generates `Trap` PDUs; these PDUs are eventually received and processed in a notification receiver application at a managing entity. The proxy forwarder application forwards request, notification, and response PDUs.

A PDU sent by an SNMP application next passes through the SNMP “engine” before it is sent via the appropriate transport protocol. Figure 9.5 shows how a PDU generated by the command generator application first enters the dispatch module,

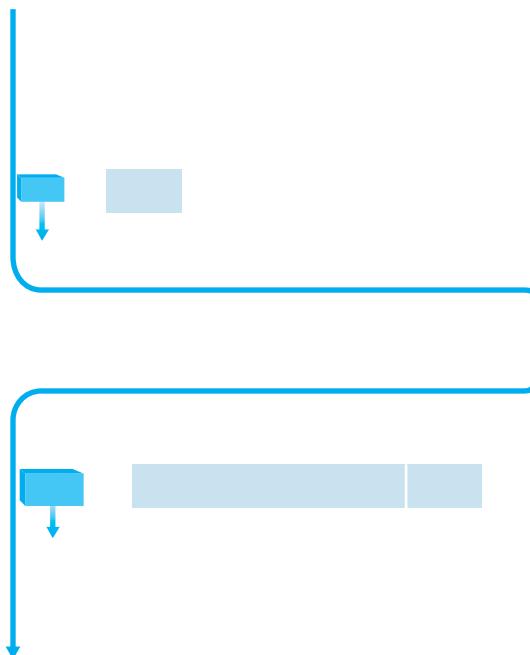


Figure 9.5 ♦ SNMPv3 engine and applications

where the SNMP version is determined. The PDU is then processed in the message-processing system, where the PDU is wrapped in a message header containing the SNMP version number, a message ID, and message size information. If encryption or authentication is needed, the appropriate header fields for this information are included as well; see [RFC 3411] for details. Finally, the SNMP message (the application-generated PDU plus the message header information) is passed to the appropriate transport protocol. The preferred transport protocol for carrying SNMP messages is UDP (that is, SNMP messages are carried as the payload in a UDP datagram), and the preferred port number for the SNMP is port 161. Port 162 is used for trap messages.

We have seen above that SNMP messages are used not just to monitor, but also to control (for example, through the `SetRequest` command) network elements. Clearly, an intruder that could intercept SNMP messages and/or generate its own SNMP packets into the management infrastructure could wreak havoc in the network. Thus, it is crucial that SNMP messages be transmitted securely. Surprisingly, it is only in the most recent version of SNMP that security has received the attention that it deserves. SNMPv3 security is known as **user-based security** [RFC 3414] in that there is the traditional concept of a user, identified by a username, with which security information such as a password, key value, or access privileges are associated. SNMPv3 provides for encryption, authentication, protection against playback attacks (see Section 8.3), and access control.

- *Encryption.* SNMP PDUs can be encrypted using the Data Encryption Standard (DES) in Cipher Block Chaining (CBC) mode. Note that since DES is a shared-key system, the secret key of the user encrypting data must be known by the receiving entity that must decrypt the data.
- *Authentication.* SNMP uses the Message Authentication Code (MAC) technique that we studied in Section 8.3.1 to provide both authentication and protection against tampering [RFC 4301]. Recall that a MAC requires the sender and receiver both to know a common secret key.
- *Protection against playback.* Recall from our discussion in Chapter 8 that nonces can be used to guard against playback attacks. SNMPv3 adopts a related approach. In order to ensure that a received message is not a replay of some earlier message, the receiver requires that the sender include a value in each message that is based on a counter in the *receiver*. This counter, which functions as a nonce, reflects the amount of time since the last reboot of the receiver's network management software and the total number of reboots since the receiver's network management software was last configured. As long as the counter in a received message is within some margin of error of the receiver's actual value, the message is accepted as a nonreplay message, at which point it may be authenticated and/or decrypted. See [RFC 3414] for details.
- *Access control.* SNMPv3 provides a view-based access control [RFC 3415] that controls which network management information can be queried and/or set by which users. An SNMP entity retains information about access rights and

policies in a Local Configuration Datastore (LCD). Portions of the LCD are themselves accessible as managed objects, defined in the View-Based Access Control Model Configuration MIB [RFC 3415], and thus can be managed and manipulated remotely via SNMP.

9.4 ASN.1

In this book, we have covered a number of interesting topics in computer networking. This section on ASN.1, however, may not make the top-ten list of interesting topics. Like vegetables, knowledge about ASN.1 and the broader issue of presentation services is something that is “good for you.” ASN.1 is an ISO-originated standard that is used in a number of Internet-related protocols, particularly in the area of network management. For example, we saw in Section 9.3 that MIB variables in SNMP were inextricably tied to ASN.1. So while the material on ASN.1 in this section may be rather dry, we hope the reader will take it on faith that the material *is* important.

In order to motivate our discussion here, consider the following thought experiment. Suppose one could reliably copy data from one computer’s memory directly into a remote computer’s memory. If one could do this, would the communication problem be “solved?” The answer to the question depends on one’s definition of “the communication problem.” Certainly, a perfect memory-to-memory copy would exactly communicate the bits and bytes from one machine to another. But does such an exact copy of the bits and bytes mean that when software running on the receiving computer accesses this data, it will see the same values that were stored into the sending computer’s memory? The answer to this question is “not necessarily!” The crux of the problem is that different computer architectures, different operating systems, and different compilers have different conventions for storing and representing data. If data is to be communicated and stored among multiple computers (as it is in every communication network), this problem of data representation must clearly be solved.

As an example of this problem, consider the simple C code fragment below. How might this structure be laid out in memory?

```
struct {
    char code;
    int x;
} test;
test.x = 259;
test.code = 'a';
```

The left side of Figure 9.6 shows a possible layout of this data on one hypothetical architecture: there is a single byte of memory containing the character **a**,

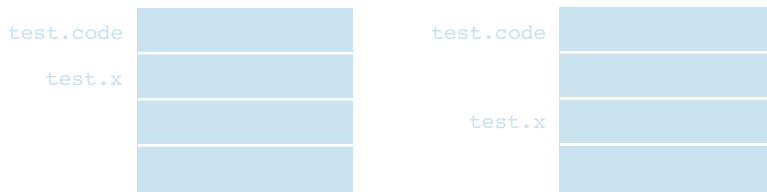


Figure 9.6 ♦ Two different data layouts on two different architectures

followed by a 16-bit word containing the integer value 259, stored with the most significant byte first. The layout in memory on another computer is shown in the right half of Figure 9.6. The character a is followed by the integer value stored with the least significant byte stored first and with the 16-bit integer aligned to start on a 16-bit word boundary. Certainly, if one were to perform a verbatim copy between these two computers' memories and use the same structure definition to access the stored values, one would see very different results on the two computers!

The fact that different architectures have different internal data formats is a real and pervasive problem. The particular problem of integer storage in different formats is so common that it has a name. “Big-endian” order for storing integers has the most significant bytes of the integer stored first (at the lowest storage address). “Little-endian” order stores the least significant bytes first. Sun SPARC and Motorola processors are big-endian, while Intel processors are little-endian. As an aside, the terms “big-endian” and “little-endian” come from the book, *Gulliver’s Travels*, by Jonathan Swift, in which two groups of people dogmatically insist on doing a simple thing in two different ways (hopefully, the analogy to the computer architecture community is clear). One group in the land of Lilliput insists on breaking their eggs at the larger end (“the big-endians”), while the other insists on breaking them at the smaller end. The difference was the cause of great civil strife and rebellion.

Given that different computers store and represent data in different ways, how should networking protocols deal with this? For example, if an SNMP agent is about to send a Response message containing the integer count of the number of received UDP datagrams, how should it represent the integer value to be sent to the managing entity—in big-endian or little-endian order? One option would be for the agent to send the bytes of the integer in the same order in which they would be stored in the managing entity. Another option would be for the agent to send in its own storage order and have the receiving entity reorder the bytes, as needed. Either option would require the sender or receiver to learn the other’s format for integer representation.

A third option is to have a machine-independent, OS-independent, language-independent method for describing integers and other data types (that is, a data-definition language) and rules that state the manner in which each of the data types is to be transmitted over the network. When data of a given type is received, it is received in a known format and can then be stored in whatever machine-specific format is required. Both the SMI that we studied in Section 9.3 and ASN.1 adopt this third option. In ISO parlance, these two standards describe a **presentation service**—the service of transmitting and translating information from one machine-specific format to another. Figure 9.7 illustrates a real-world presentation problem; neither receiver understands the essential idea being communicated—that the speaker likes something. As shown in Figure 9.8, a presentation service can solve this problem by translating the idea into a commonly understood (by the presentation service), person-independent language, sending that information to the receiver, and then translating into a language understood by the receiver.

Table 9.5 shows a few of the ASN.1-defined data types. Recall that we encountered the INTEGER, OCTET STRING, and OBJECT IDENTIFIER data types in our earlier study of the SMI. Since our goal here is (mercifully) not to provide a complete introduction to ASN.1, we refer the reader to the standards or to the printed and online book [Larmouth 1996] for a description of ASN.1 types and constructors, such as SEQUENCE and SET, that allow for the definition of structured data types.

In addition to providing a data definition language, ASN.1 also provides **Basic Encoding Rules (BER)** that specify how instances of objects that have been defined using the ASN.1 data definition language are to be sent over the network. The BER adopts a so-called **TLV (Type, Length, Value) approach** to encoding data for transmission. For each data item to be sent, the data type, the length of the data item,

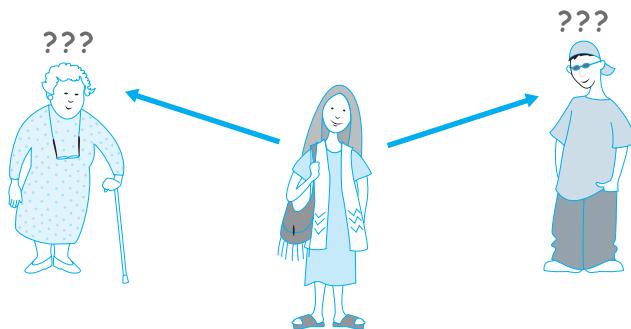


Figure 9.7 ♦ The presentation problem

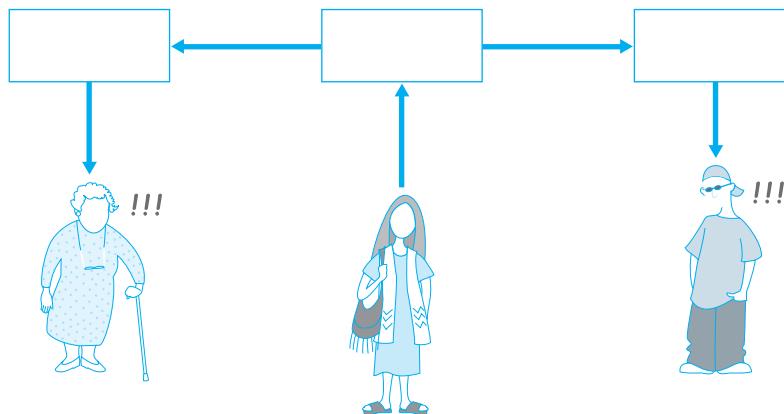


Figure 9.8 ♦ The presentation problem solved

and then the actual value of the data item are sent, in that order. With this simple convention, the received data is essentially self-identifying.

Figure 9.9 shows how the two data items in a simple example would be sent. In this example, the sender wants to send the character string “smith” followed by the value 259 decimal (which equals 00000001 00000011 in binary, or a byte value of 1 followed by a byte value of 3), assuming big-endian order. The first byte in the

Tag	Type	Description
1	BOOLEAN	value is “true” or “false”
2	INTEGER	can be arbitrarily large
3	BITSTRING	list of one or more bits
4	OCTET STRING	list of one or more bytes
5	NULL	no value
6	OBJECT IDENTIFIER	name, in the ASN.1 standard naming tree; see Section 9.2.2
9	REAL	floating point

Table 9.5 ♦ Selected ASN.1 data types

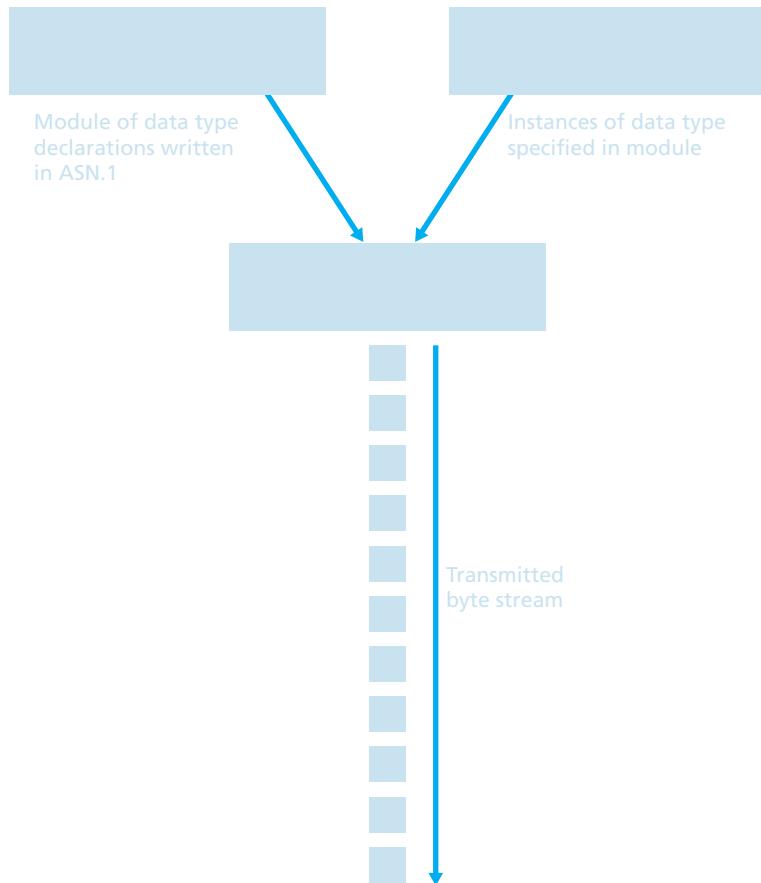


Figure 9.9 ♦ BER encoding example

transmitted stream has the value 4, indicating that the type of the following data item is an OCTET STRING; this is the “T” in the TLV encoding. The second byte in the stream contains the length of the OCTET STRING, in this case 5. The third byte in the transmitted stream begins the OCTET STRING of length 5; it contains the ASCII representation of the letter *s*. The T, L, and V values of the next data item are 2 (the INTEGER type tag value), 2 (that is, an integer of length 2 bytes), and the 2-byte big-endian representation of the value 259 decimal.

In our previous discussion, we have only touched on a small and simple subset of ASN.1. Resources for learning more about ASN.1 include the ASN.1 standards document [ISO X.680 2002], the online OSI-related book [Larmouth 2012], and the ASN.1-related Web sites, [OSS 2012] and [OID Repository 2012].

9.5 Conclusion

Our study of network management, and indeed of all of networking, is now complete!

In this final chapter on network management, we began by motivating the need for providing appropriate tools for the network administrator—the person whose job it is to keep the network “up and running”—for monitoring, testing, polling, configuring, analyzing, evaluating, and controlling the operation of the network. Our analogies with the management of complex systems such as power plants, airplanes, and human organization helped motivate this need. We saw that the architecture of network management systems revolves around five key components: (1) a network manager, (2) a set of managed remote (from the network manager) devices, (3) the Management Information Bases (MIBs) at these devices, containing data about the devices’ status and operation, (4) remote agents that report MIB information and take action under the control of the network manager, and (5) a protocol for communication between the network manager and the remote devices.

We then delved into the details of the Internet-Standard Management Framework, and the SNMP protocol in particular. We saw how SNMP instantiates the five key components of a network management architecture, and we spent considerable time examining MIB objects, the SMI—the data definition language for specifying MIBs, and the SNMP protocol itself. Noting that the SMI and ASN.1 are inextricably tied together, and that ASN.1 plays a key role in the presentation layer in the ISO/OSI seven-layer reference model, we then briefly examined ASN.1. Perhaps more important than the details of ASN.1 itself was the noted need to provide for translation between machine-specific data formats in a network. While some network architectures explicitly acknowledge the importance of this service by having a presentation layer, this layer is absent in the Internet protocol stack.

It is also worth noting that there are many topics in network management that we chose *not* to cover—topics such as fault identification and management, proactive anomaly detection, alarm correlation, and the larger issues of service management (for example, as opposed to network management). While important, these topics would form a text in their own right, and we refer the reader to the references noted in Section 9.1.



Homework Problems and Questions

Chapter 9 Review Questions

SECTION 9.1

- R1. Why would a network manager benefit from having network management tools? Describe five scenarios.
- R2. What are the five areas of network management defined by the ISO?

- R3. What is the difference between network management and service management?

SECTION 9.2

- R4. Define the following terms: managing entity, managed device, management agent, MIB, network management protocol.

SECTION 9.3

- R5. What is the role of the SMI in network management?
- R6. What is an important difference between a request-response message and a trap message in SNMP?
- R7. What are the seven message types used in SNMP?
- R8. What is meant by an “SNMP engine”?

SECTION 9.4

- R9. What is the purpose of the ASN.1 object identifier tree?
- R10. What is the role of ASN.1 in the ISO/OSI reference model’s presentation layer?
- R11. Does the Internet have a presentation layer? If not, how are concerns about differences in machine architectures—for example, the different representation of integers on different machines—addressed?
- R12. What is meant by TLV encoding?



Problems

- P1. Consider the two ways in which communication occurs between a managing entity and a managed device: request-response mode and trapping. What are the pros and cons of these two approaches, in terms of (1) overhead, (2) notification time when exceptional events occur, and (3) robustness with respect to lost messages between the managing entity and the device?
- P2. In Section 9.3 we saw that it was preferable to transport SNMP messages in unreliable UDP datagrams. Why do you think the designers of SNMP chose UDP rather than TCP as the transport protocol of choice for SNMP?
- P3. What is the ASN.1 object identifier for the ICMP protocol (see Figure 9.3)?
- P4. Suppose you worked for a US-based company that wanted to develop its own MIB for managing a product line. Where in the object identifier tree (Figure 9.3) would it be registered? (*Hint:* You’ll have to do some digging through RFCs or other documents to answer this question.)

- P5. Recall from Section 9.3.2 that a private company (enterprise) can create its own MIB variables under the private branch 1.3.6.4. Suppose that IBM wanted to create a MIB for its Web server software. What would be the next OID qualifier after 1.3.6.1.4? (In order to answer this question, you will need to consult [IANA 2009b]). Search the Web and see if you can find out whether such a MIB exists for an IBM server.
- P6. Why do you think the length precedes the value in a TLV encoding (rather than the length following the value)?
- P7. Consider Figure 9.9. What would be the BER encoding of {`weight`, 165} {`lastname`, "Michael"}?
- P8. Consider Figure 9.9. What would be the BER encoding of {`weight`, 145} {`lastname`, "Sridhar"}?

AN INTERVIEW WITH...

Jennifer Rexford

Jennifer Rexford is a Professor in the Computer Science department at Princeton University. Her research has the broad goal of making computer networks easier to design and manage, with particular emphasis on routing protocols. From 1996–2004, she was a member of the Network Management and Performance department at AT&T Labs–Research. While at AT&T, she designed techniques and tools for network measurement, traffic engineering, and router configuration that were deployed in AT&T’s backbone network. Jennifer is co-author of the book “Web Protocols and Practice: Networking Protocols, Caching, and Traffic Measurement,” published by Addison-Wesley in May 2001. She served as the chair of ACM SIGCOMM from 2003 to 2007. She received her BSE degree in electrical engineering from Princeton University in 1991, and her MSE and PhD degrees in electrical engineering and computer science from the University of Michigan in 1993 and 1996, respectively. In 2004, Jennifer was the winner of ACM’s Grace Murray Hopper Award for outstanding young computer professional and appeared on the MIT TR-100 list of top innovators under the age of 35.



Please describe one or two of the most exciting projects you have worked on during your career. What were the biggest challenges?

When I was a researcher at AT&T, a group of us designed a new way to manage routing in Internet Service Provider backbone networks. Traditionally, network operators configure each router individually, and these routers run distributed protocols to compute paths through the network. We believed that network management would be simpler and more flexible if network operators could exercise *direct* control over how routers forward traffic based on a *network-wide* view of the topology and traffic. The Routing Control Platform (RCP) we designed and built could compute the routes for all of AT&T’s backbone on a single commodity computer, and could control legacy routers without modification. To me, this project was exciting because we had a provocative idea, a working system, and ultimately a real deployment in an operational network.

What changes and innovations do you see happening in network management in the future?

Rather than simply “bolting on” network management on top of existing networks, researchers and practitioners alike are starting to design networks that are fundamentally easier to manage. Like our early work on the RCP, the main idea in so-called Software Defined Networking (SDN) is to run a controller that can install low-level packet-handling rules in the underlying switches using a standard protocol. This controller can run various

network-management applications, such as dynamic access control, seamless user mobility, traffic engineering, server load balancing, energy-efficient networking, and so on. I believe SDN is a great opportunity to get network management right, by rethinking the relationship between the network devices and the software that manages them.

Where do you see the future of networking and the Internet?

Networking is an exciting field because the applications and the underlying technologies change all the time. We are always reinventing ourselves! Who would have predicted even five or ten years ago the dominance of smart phones, allowing mobile users to access existing applications as well as new location-based services? The emergence of cloud computing is fundamentally changing the relationship between users and the applications they run, and networked sensors are enabling a wealth of new applications. The pace of innovation is truly inspiring.

The underlying network is a crucial component in all of these innovations. Yet, the network is notoriously “in the way”—limiting performance, compromising reliability, constraining applications, and complicating the deployment and management of services. We should strive to make the network of the future as invisible as the air we breathe, so it never stands in the way of new ideas and valuable services. To do this, we need to raise the level of abstraction above individual network devices and protocols (and their attendant acronyms!), so we can reason about the network as a whole.

What people inspired you professionally?

I've long been inspired by Sally Floyd at the International Computer Science Institute. Her research is always purposeful, focusing on the important challenges facing the Internet. She digs deeply into hard questions until she understands the problem and the space of solutions completely, and she devotes serious energy into “making things happen,” such as pushing her ideas into protocol standards and network equipment. Also, she gives back to the community, through professional service in numerous standards and research organizations and by creating tools (such as the widely used ns-2 and ns-3 simulators) that enable other researchers to succeed. She retired in 2009 but her influence on the field will be felt for years to come.

What are your recommendations for students who want careers in computer science and networking?

Networking is an inherently interdisciplinary field. Applying techniques from other disciplines to networking problems is a great way to move the field forward. We've seen tremendous

breakthroughs in networking come from such diverse areas as queuing theory, game theory, control theory, distributed systems, network optimization, programming languages, machine learning, algorithms, data structures, and so on. I think that becoming conversant in a related field, or collaborating closely with experts in those fields, is a wonderful way to put networking on a stronger foundation, so we can learn how to build networks that are worthy of society's trust. Beyond the theoretical disciplines, networking is exciting because we create real artifacts that real people use. Mastering how to design and build systems—by gaining experience in operating systems, computer architecture, and so on—is another fantastic way to amplify your knowledge of networking to help change the world.

References

A note on URLs. In the references below, we have provided URLs for Web pages, Web-only documents, and other material that has not been published in a conference or journal (when we have been able to locate a URL for such material). We have not provided URLs for conference and journal publications, as these documents can usually be located via a search engine, from the conference Web site (e.g., papers in all *ACM SIGCOMM* conferences and workshops can be located via <http://www.acm.org/sigcomm>), or via a digital library subscription. While all URLs provided below were valid (and tested) in Jan. 2012, URLs can become out of date. Please consult the online version of this book (<http://www.awl.com/kurose-ross>) for an up-to-date bibliography.

A note on Internet Request for Comments (RFCs): Copies of Internet RFCs are available at many sites. The RFC Editor of the Internet Society (the body that oversees the RFCs) maintains the site, <http://www.rfc-editor.org>. This site allows you to search for a specific RFC by title, number, or authors, and will show updates to any RFCs listed. Internet RFCs can be updated or obsoleted by later RFCs. Our favorite site for getting RFCs is the original source—<http://www.rfc-editor.org>.

[3Com Addressing 2012] 3Com Corp., “White paper: Understanding IP addressing: Everything you ever wanted to know,” http://www.3com.com/other/pdfs/infra/corpinfo/en_US/501302.pdf

[3GPP 2012] Third Generation Partnership Project homepage, <http://www.3gpp.org/>

[3GPP Network Architecture 2012] 3GPP, “TS 23.002: Network Architecture: Digital Cellular Telecommunications System (Phase 2+); Universal Mobile Telecommunications System (UMTS); LTE,” <http://www.3gpp.org/ftp/Specs/html-info/23002.htm>

[Albitz 1993] P. Albitz and C. Liu, *DNS and BIND*, O’Reilly & Associates, Petaluma, CA, 1993.

[Abramson 1970] N. Abramson, “The Aloha System—Another Alternative for Computer Communications,” *Proc. 1970 Fall Joint Computer Conference, AFIPS Conference*, p. 37, 1970.

[Abramson 1985] N. Abramson, “Development of the Alohanet,” *IEEE Transactions on Information Theory*, Vol. IT-31, No. 3 (Mar. 1985), pp. 119–123.

[Abramson 2009] N. Abramson, “The Alohanet – Surfing for Wireless Data,” *IEEE Communications Magazine*, Vol. 47, No. 12, pp. 21–25.

[Abu-Libdeh 2010] H. Abu-Libdeh, P. Costa, A. Rowstron, G. O’Shea, A. Donnelly, “Symbiotic Routing in Future Data Centers,” *Proc. 2010 ACM SIGCOMM*.

[Adhikari 2011a] V. K. Adhikari, S. Jain, Y. Chen, Z. L. Zhang, “Vivisecting YouTube: An Active Measurement Study,” Technical Report, University of Minnesota, 2011.

[Adhikari 2012] V. K. Adhikari, Y. Gao, F. Hao, M. Varvello, V. Hilt, M. Steiner, Z. L. Zhang, “Unreeling Netflix: Understanding and Improving Multi-CDN Movie Delivery,” Technical Report, University of Minnesota, 2012.

[Afanasyev 2010] A. Afanasyev, N. Tilley, P. Reiher, L. Kleinrock, “Host-to-Host Congestion Control for TCP,” *IEEE Communications Surveys & Tutorials*, Vol. 12, No. 3, pp. 304–342.

- [Agarwal 2009] S. Agarwal, J. Lorch, “Matchmaking for Online Games and Other Latency-sensitive P2P Systems,” *Proc. 2009 ACM SIGCOMM*.
- [Ahn 1995] J. S. Ahn, P. B. Danzig, Z. Liu, and Y. Yan, “Experience with TCP Vegas: Emulation and Experiment,” *Proc. 1995 ACM SIGCOMM* (Boston, MA, Aug. 1995), pp. 185–195.
- [Akamai 2012] Akamai homepage, <http://www.akamai.com>
- [Akella 2003] A. Akella, S. Seshan, A. Shaikh, “An Empirical Evaluation of Wide-Area Internet Bottlenecks,” *Proc. 2003 ACM Internet Measurement Conference* (Miami, FL, Nov. 2003).
- [Akhshabi 2011] S. Akhshabi, A. C. Begen, C. Dovrolis, “An Experimental Evaluation of Rate-Adaptation Algorithms in Adaptive Streaming over HTTP,” *Proc. 2011 ACM Multimedia Systems Conf.*
- [Akyildiz 2010] I. Akyildiz, D. Gutierrez-Estevez, E. Reyes, “The Evolution to 4G Cellular Systems, LTE Advanced,” *Physical Communication*, Elsevier, 3 (2010), 217–244.
- [Alcatel-Lucent 2009] Alcatel-Lucent, “Introduction to Evolved Packet Core,” http://downloads.lightreading.com/wplib/alcatellucent/ALU_WP_Intro_to_EPC.pdf
- [Al-Fares 2008] M. Al-Fares, A. Loukissas, A. Vahdat, “A Scalable, Commodity Data Center Network Architecture,” *Proc. 2008 ACM SIGCOMM*.
- [Alizadeh 2010] M. Alizadeh, A. Greenberg, D. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, M. Sridharan, “Data Center TCP (DCTCP),” *Proc. 2010 ACM SIGCOMM*.
- [Allman 2011] E. Allman, “The Robustness Principle Reconsidered: Seeking a Middle Ground,” *Communications of the ACM*, Vol. 54, No. 8 (Aug. 2011), pp. 40–45.
- [Anderson 1995] J. B. Andersen, T. S. Rappaport, S. Yoshida, “Propagation Measurements and Models for Wireless Communications Channels,” *IEEE Communications Magazine*, (Jan. 1995), pp. 42–49.
- [Andrews 2002] M. Andrews, M. Shepherd, A. Srinivasan, P. Winkler, F. Zane, “Clustering and Server Election Using Passive Monitoring,” *Proc. 2002 IEEE INFOCOM*.
- [Androulidakis-Theotokis 2004] S. Androulidakis-Theotokis, D. Spinellis, “A Survey of Peer-to-Peer Content Distribution Technologies,” *ACM Computing Surveys*, Vol. 36, No. 4 (Dec. 2004), pp. 335–371.
- [Aperjis 2008] C. Aperjis, M.J. Freedman, R. Johari, “Peer-Assisted Content Distribution with Prices,” *Proc. ACM CoNEXT’08* (Madrid, Dec. 2008).
- [Appenzeller 2004] G. Appenzeller, I. Keslassy, N. McKeown, “Sizing Router Buffers,” *Proc. 2004 ACM SIGCOMM* (Portland, OR, Aug. 2004).
- [Ash 1998] G. R. Ash, *Dynamic Routing in Telecommunications Networks*, McGraw Hill, New York, NY, 1998.
- [ASO-ICANN 2012] The Address Supporting Organization home page, <http://www.aso.icann.org>
- [AT&T SLA 2012] AT&T, “AT&T High Speed Internet Business Edition Service Level Agreements,” <http://www.att.com/gen/general?pid=6622>
- [Atheros 2012] Atheros Communications Inc., “Atheros AR5006 WLAN Chipset Product Bulletins,” <http://www.atheros.com/pt/AR5006Bulletins.htm>
- [Augustin 2009] B. Augustin, B. Krishnamurthy, W. Willinger, “IXPs: Mapped?” *Proc. Internet Measurement Conference (IMC)*, November 2009.

- [Ayanoglu 1995] E. Ayanoglu, S. Paul, T. F. La Porta, K. K. Sabnani, R. D. Gitlin, “AIRMAIL: A Link-Layer Protocol for Wireless Networks,” *ACM ACM/Baltzer Wireless Networks Journal*, 1: 47–60, Feb. 1995.
- [Bakre 1995] A. Bakre, B. R. Badrinath, “I-TCP: Indirect TCP for Mobile Hosts,” *Proc. 1995 Int. Conf. on Distributed Computing Systems (ICDCS)* (May 1995), pp. 136–143.
- [Balakrishnan 1997] H. Balakrishnan, V. Padmanabhan, S. Seshan, R. Katz, “A Comparison of Mechanisms for Improving TCP Performance Over Wireless Links,” *IEEE/ACM Transactions on Networking* Vol. 5, No. 6 (Dec. 1997).
- [Balakrishnan 2003] H. Balakrishnan, F. Kaashoek, D. Karger, R. Morris, I. Stoica, “Looking Up Data in P2P Systems,” *Communications of the ACM*, Vol. 46, No. 2 (Feb. 2003), pp. 43–48.
- [Baldauf 2007] M. Baldauf, S. Dustdar, F. Rosenberg, “A Survey on Context-Aware Systems,” *Int. J. Ad Hoc and Ubiquitous Computing*, Vol. 2, No. 4 (2007), pp. 263–277.
- [Ballani 2006] H. Ballani, P. Francis, S. Ratnasamy, “A Measurement-based Deployment Proposal for IP Anycast,” *Proc. 2006 ACM Internet Measurement Conf.*
- [Ballani 2011] H. Ballani, P. Costa, T. Karagiannis, Ant Rowstron, “Towards Predictable Datacenter Networks,” *Proc. 2011 ACM SIGCOMM*.
- [Baran 1964] P. Baran, “On Distributed Communication Networks,” *IEEE Transactions on Communication Systems*, Mar. 1964. Rand Corporation Technical report with the same title (Memorandum RM-3420-PR, 1964). <http://www.rand.org/publications/RM/RM3420/>
- [Bardwell 2004] J. Bardwell, “You Believe You Understand What You Think I Said . . . The Truth About 802.11 Signal And Noise Metrics: A Discussion Clarifying Often-Misused 802.11 WLAN Terminologies,” http://www.connect802.com/download/techpubs/2004/you_believe_D100201.pdf
- [Barford 2009] P. Barford, N. Duffield, A. Ron, J. Sommers, “Network Performance Anomaly Detection and Localization,” *Proc. 2009 IEEE INFOCOM* (Apr. 2009).
- [Baronti 2007] P. Baronti, P. Pillai, V. Chook, S. Chessa, A. Gotta, Y. Hu, “Wireless Sensor Networks: A Survey on the State of the Art and the 802.15.4 and ZigBee Standards,” *Computer Communications*, Vol. 30, No. 7 (2007), pp. 1655–1695.
- [Baset 2006] S. A. Basset and H. Schulzrinne, “An analysis of the Skype peer-to-peer Internet Telephony Protocol,” *Proc. 2006 IEEE INFOCOM* (Barcelona, Spain, Apr. 2006).
- [BBC 2001] BBC news online “A Small Slice of Design,” Apr. 2001, <http://news.bbc.co.uk/2/hi/science/nature/1264205.stm>
- [BBC 2012] BBC, “Multicast,” <http://www.bbc.co.uk/multicast/>
- [Beheshti 2008] N. Beheshti, Y. Ganjali, M. Ghobadi, N. McKeown, G. Salmon, “Experimental Study of Router Buffer Sizing,” *Proc. ACM Internet Measurement Conference* (October 2008, Vouliagmeni, Greece).
- [Bender 2000] P. Bender, P. Black, M. Grob, R. Padovani, N. Sindhushayana, A. Viterbi, “CDMA/HDR: A bandwidth-efficient high-speed wireless data service for nomadic users,” *IEEE Commun. Mag.*, Vol. 38, No. 7 (July 2000) pp. 70–77.
- [Berners-Lee 1989] T. Berners-Lee, CERN, “Information Management: A Proposal,” Mar. 1989, May 1990. <http://www.w3.org/History/1989/proposal.html>
- [Berners-Lee 1994] T. Berners-Lee, R. Cailliau, A. Luotonen, H. Frystyk Nielsen, A. Secret, “The World-Wide Web,” *Communications of the ACM*, Vol. 37, No. 8 (Aug. 1994), pp. 76–82.

- [Bertsekas 1991]** D. Bertsekas, R. Gallagher, *Data Networks*, 2nd Ed., Prentice Hall, Englewood Cliffs, NJ, 1991.
- [Biddle 2003]** P. Biddle, P. England, M. Peinado, B. Willman, “The Darknet and the Future of Content Distribution,” *2002 ACM Workshop on Digital Rights Management*, (Nov. 2002, Washington, D.C.) <http://crypto.stanford.edu/DRM2002/darknet5.doc>
- [Biersack 1992]** E. W. Biersack, “Performance evaluation of forward error correction in ATM networks,” *Proc. 1999 ACM SIGCOMM* (Baltimore, MD, Aug. 1992), pp. 248–257.
- [BIND 2012]** Internet Software Consortium page on BIND, <http://www.isc.org/bind.html>
- [Bisdikian 2001]** C. Bisdikian, “An Overview of the Bluetooth Wireless Technology,” *IEEE Communications Magazine*, No. 12 (Dec. 2001), pp. 86–94.
- [Bishop 2003]** M. Bishop, *Computer Security: Art and Science*, Boston: Addison Wesley, Boston MA, 2003.
- [Black 1995]** U. Black, *ATM Volume I: Foundation for Broadband Networks*, Prentice Hall, 1995.
- [Black 1997]** U. Black, *ATM Volume II: Signaling in Broadband Networks*, Prentice Hall, 1997.
- [Blumenthal 2001]** M. Blumenthal, D. Clark, “Rethinking the Design of the Internet: the End-to-end Arguments vs. the Brave New World,” *ACM Transactions on Internet Technology*, Vol. 1, No. 1 (Aug. 2001), pp. 70–109.
- [Bochman 1984]** G. V. Bochmann, C. A. Sunshine, “Formal methods in communication protocol design,” *IEEE Transactions on Communications*, Vol. 28, No. 4 (Apr. 1980) pp. 624–631.
- [Bolot 1994]** J-C. Bolot, T. Turletti, “A rate control scheme for packet video in the Internet,” *Proc. 1994 IEEE INFOCOM*, pp. 1216–1223.
- [Bolot 1996]** J-C. Bolot, A. Vega-Garcia, “Control Mechanisms for Packet Audio in the Internet,” *Proc. 1996 IEEE INFOCOM*, pp. 232–239.
- [Bradner 1996]** S. Bradner, A. Mankin, *IPng: Internet Protocol Next Generation*, Addison-Wesley, Reading, MA, 1996.
- [Brakmo 1995]** L. Brakmo, L. Peterson, “TCP Vegas: End to End Congestion Avoidance on a Global Internet,” *IEEE Journal of Selected Areas in Communications*, Vol. 13, No. 8 (Oct. 1995), pp. 1465–1480.
- [Breslau 2000]** L. Breslau, E. Knightly, S. Shenker, I. Stoica, H. Zhang, “Endpoint Admission Control: Architectural Issues and Performance,” *Proc. 2000 ACM SIGCOMM* (Stockholm, Sweden, Aug. 2000).
- [Bryant 1988]** B. Bryant, “Designing an Authentication System: A Dialogue in Four Scenes,” <http://web.mit.edu/kerberos/www/dialogue.html>
- [Bush 1945]** V. Bush, “As We May Think,” *The Atlantic Monthly*, July 1945. <http://www.theatlantic.com/unbound/flashbks/computer/bushf.htm>
- [Byers 1998]** J. Byers, M. Luby, M. Mitzenmacher, A. Rege, “A digital fountain approach to reliable distribution of bulk data,” *Proc. 1998 ACM SIGCOMM* (Vancouver, Canada, Aug. 1998), pp. 56–67.
- [Cablelabs 2012]** CableLabs homepage, <http://www.cablelabs.com>
- [CacheLogic 2012]** CacheLogic homepage, <http://www.cachelogic.com>

- [Caesar 2005a] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, J. van der Merwe, “Design and implementation of a Routing Control Platform,” *Proc. Networked Systems Design and Implementation* (May 2005).
- [Caesar 2005b] M. Caesar, J. Rexford, “BGP Routing Policies in ISP Networks,” *IEEE Network Magazine*, Vol. 19, No. 6 (Nov. 2005).
- [Casado 2009] M. Casado, M. Freedman, J. Pettit, J. Luo, N. Gude, N. McKeown, S. Shenker, “Rethinking Enterprise Network Control,” *IEEE/ACM Transactions on Networking (ToN)*, Vol. 17, No. 4 (Aug. 2009), pp. 1270–1283.
- [Caldwell 2012] C. Caldwell, “The Prime Pages,” <http://www.utm.edu/research/primes/prove>
- [Cardwell 2000] N. Cardwell, S. Savage, T. Anderson, “Modeling TCP Latency,” *Proc. 2000 IEEE INFOCOM* (Tel-Aviv, Israel, Mar. 2000).
- [CASA 2012] Center for Collaborative Adaptive Sensing of the Atmosphere, <http://www.casa.umass.edu>
- [Casado 2007] M. Casado, M. Freedman, J. Pettit, J. Luo, N. McKeown, S. Shenker, “Ethane: Taking Control of the Enterprise,” *Proc. 2007 ACM SIGCOMM* (Kyoto, Japan, Aug. 2007).
- [Casner 1992] S. Casner, S. Deering, “First IETF Internet Audiocast,” *ACM SIGCOMM Computer Communications Review*, Vol. 22, No. 3 (July 1992), pp. 92–97.
- [Ceiva 2012] Ceiva homepage, <http://www.ceiva.com/>
- [CENS 2012] Center for Embedded Network Sensing, <http://www.cens.ucla.edu/>
- [Cerf 1974] V. Cerf, R. Kahn, “A Protocol for Packet Network Interconnection,” *IEEE Transactions on Communications Technology*, Vol. COM-22, No. 5, pp. 627–641.
- [CERT 2001–09] CERT, “Advisory 2001–09: Statistical Weaknesses in TCP/IP Initial Sequence Numbers,” <http://www.cert.org/advisories/CA-2001-09.html>
- [CERT 2003–04] CERT, “CERT Advisory CA-2003-04 MS-SQL Server Worm,” <http://www.cert.org/advisories/CA-2003-04.html>
- [CERT 2012] CERT Coordination Center, <http://www.cert.org/advisories>
- [CERT Filtering 2012] CERT, “Packet Filtering for Firewall Systems,” http://www.cert.org/tech_tips/packet_filtering.html
- [Cert SYN 1996] CERT, “Advisory CA-96.21: TCP SYN Flooding and IP Spoofing Attacks,” <http://www.cert.org/advisories/CA-1998-01.html>
- [Chao 2001] H. J. Chao, C. Lam, E. Oki, *Broadband Packet Switching Technologies—A Practical Guide to ATM Switches and IP Routers*, John Wiley & Sons, 2001.
- [Chao 2011] C. Zhang, P. Dunghel, D. Wu, K. W. Ross, “Unraveling the BitTorrent Ecosystem,” *IEEE Transactions on Parallel and Distributed Systems*, Vol. 22, No. 7 (July 2011).
- [Chen 2000] G. Chen, D. Kotz, “A Survey of Context-Aware Mobile Computing Research,” *Technical Report TR2000-381*, Dept. of Computer Science, Dartmouth College, Nov. 2000. <http://www.cs.dartmouth.edu/reports/TR2000-381.pdf>
- [Chen 2006] K.-T. Chen, C.-Y. Huang, P. Huang, C.-L. Lei, “Quantifying Skype User Satisfaction,” *Proc. 2006 ACM SIGCOMM* (Pisa, Italy, Sept. 2006).
- [Chen 2010] K. Chen, C. Guo, H. Wu, J. Yuan, Z. Feng, Y. Chen, S. Lu, W. Wu, “Generic and Automatic Address Configuration for Data Center Networks,” *Proc. 2010 ACM SIGCOMM*.

- [Chen 2011]** Y. Chen, S. Jain, V. K. Adhikari, Z. Zhang, “Characterizing Roles of Front-End Servers in End-to-End Performance of Dynamic Content Distribution,” *Proc. 2011 ACM Internet Measurement Conference* (Berlin, Germany, Nov. 2011).
- [Chenoweth 2010]** T. Chenoweth, R. Minch, S. Tabor, “Wireless Insecurity: Examining User Security Behavior on Public Networks,” *Communications of the ACM*, Vol. 53, No. 2 (Feb. 2010), pp. 134–138.
- [Cheswick 2000]** B. Cheswick, H. Burch, S. Branigan, “Mapping and Visualizing the Internet,” *Proc. 2000 Usenix Conference* (San Diego, CA, June 2000).
- [Chiu 1989]** D. Chiu, R. Jain, “Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks,” *Computer Networks and ISDN Systems*, Vol. 17, No. 1, pp. 1–14. http://www.cs.wustl.edu/~jain/papers/cong_av.htm
- [Christiansen 2001]** M. Christiansen, K. Jeffay, D. Ott, F. D. Smith, “Tuning Red for Web Traffic,” *IEEE/ACM Transactions on Networking*, Vol. 9, No. 3 (June 2001), pp. 249–264.
- [Chu 2002]** Y. Chu, S. Rao, S. Seshan, H. Zhang, “A Case for End System Multicast,” *IEEE J. Selected Areas in Communications*, Vol 20, No. 8 (Oct. 2002), pp. 1456–1471.
- [Chuang 2005]** S. Chuang, S. Iyer, N. McKeown, “Practical Algorithms for Performance Guarantees in Buffered Crossbars,” *Proc. 2005 IEEE INFOCOM*.
- [Cicconetti 2006]** C. Cicconetti, L. Lenzini, A. Mingozi, K. Eklund, “Quality of Service Support in 802.16 Networks,” *IEEE Network Magazine* (Mar./Apr. 2006), pp. 50–55.
- [Cisco 12000 2012]** Cisco Systems Inc., “Cisco XR 12000 Series and Cisco 12000 Series Routers,” <http://www.cisco.com/en/US/products/ps6342/index.html>
- [Cisco 8500 2012]** Cisco Systems Inc., “Catalyst 8500 Campus Switch Router Architecture,” http://www.cisco.com/univercd/cc/td/doc/product/l3sw/8540/rel_12_0/w5_6f/softcnfg/lcfg8500.pdf
- [Cisco 2011]** Cisco Visual Networking Index: Forecast and Methodology, 2010–2015, White Paper, 2011.
- [Cisco 2012]** Cisco 2012, Data Centers, <http://www.cisco.com/go/dce>
- [Cisco NAT 2012]** Cisco Systems Inc., “How NAT Works,” http://www.cisco.com/en/US/tech/tk648/tk361/technologies_tech_note09186a0080094831.shtml
- [Cisco QoS 2012]** Cisco Systems Inc., “Advanced QoS Services for the Intelligent Internet,” http://www.cisco.com/warp/public/cc/pd/iosw/loft/loqo/tech/qos_wp.htm
- [Cisco Queue 2012]** Cisco Systems Inc., “Congestion Management Overview,” http://www.cisco.com/en/US/docs/ios/12_2/qos/configuration/guide/qcfconmg.html
- [Cisco Switches 2012]** Cisco Systems Inc., “Multiservice Switches,” <http://www.cisco.com/warp/public/cc/pd/si/index.shtml>
- [Cisco SYN 2012]** Cisco Systems Inc., “Defining Strategies to Protect Against TCP SYN Denial of Service Attacks,” http://www.cisco.com/en/US/tech/tk828/technologies_tech_note09186a00800f67d5.shtml
- [Cisco VNI 2011]** Cisco, “Visual Networking Index,” http://www.cisco.com/web/solutions/sp/vni/vni_forecast_highlights/index.html
- [Clark 1988]** D. Clark, “The Design Philosophy of the DARPA Internet Protocols,” *Proc. 1988 ACM SIGCOMM* (Stanford, CA, Aug. 1988).
- [Clarke 2002]** I. Clarke, T. W. Hong, S. G. Miller, O. Sandberg, B. Wiley, “Protecting Free Expression Online with Freenet,” *IEEE Internet Computing* (Jan.–Feb. 2002), pp. 40–49.

- [**Cohen 1977**] D. Cohen, “Issues in Transnet Packetized Voice Communication,” *Proc. Fifth Data Communications Symposium* (Snowbird, UT, Sept. 1977), pp. 6–13.
- [**Cohen 2003**] B. Cohen, “Incentives to Build Robustness in BitTorrent,” *First Workshop on the Economics of Peer-to-Peer Systems* (Berkeley, CA, June 2003).
- [**Cookie Central 2012**] Cookie Central homepage, http://www.cookiecentral.com/n_cookie_faq.htm
- [**CoolStreaming 2005**] X. Zhang, J. Liu, J., B. Li, and T.-S. P. Yum, “CoolStreamingDONet: A Data-driven Overlay Network for Peer-to-Peer Live Media Streaming,” *Proc. 2005 IEEE INFOCOM* (Miami, FL, Mar. 2005).
- [**Cormen 2001**] T. H. Cormen, *Introduction to Algorithms*, 2nd Ed., MIT Press, Cambridge, MA, 2001.
- [**Crow 1997**] B. Crow, I. Widjaja, J. Kim, P. Sakai, “IEEE 802.11 Wireless Local Area Networks,” *IEEE Communications Magazine* (Sept. 1997), pp. 116–126.
- [**Crowcroft 1995**] J. Crowcroft, Z. Wang, A. Smith, J. Adams, “A Comparison of the IETF and ATM Service Models,” *IEEE Communications Magazine* (Nov./Dec. 1995), pp. 12–16.
- [**Crowcroft 1999**] J. Crowcroft, M. Handley, I. Wakeman, *Internetworking Multimedia*, Morgan-Kaufman, San Francisco, 1999.
- [**Curtis 2011**] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, S. Banerjee, “DevoFlow: Scaling Flow Management for High-Performance Networks,” *Proc. 2011 ACM SIGCOMM*.
- [**Cusumano 1998**] M. A. Cusumano, D. B. Yoffie, *Competing on Internet Time: Lessons from Netscape and its Battle with Microsoft*, Free Press, New York, NY, 1998.
- [**Dahlman 1998**] E. Dahlman, B. Gudmundson, M. Nilsson, J. Sköld, “UMTS/IMT-2000 Based on Wideband CDMA,” *IEEE Communications Magazine* (Sept. 1998), pp. 70–80.
- [**Daigle 1991**] J. N. Daigle, *Queueing Theory for Telecommunications*, Addison-Wesley, Reading, MA, 1991.
- [**Dalal 1978**] Y. Dalal, R. Metcalfe, “Reverse Path Forwarding of Broadcast Packets,” *Communications of the ACM*, Vol. 21, No. 12 (Dec. 1978), pp. 1040–1048.
- [**Davie 2000**] B. Davie and Y. Rekhter, *MPLS: Technology and Applications*, Morgan Kaufmann Series in Networking, 2000.
- [**Davies 2005**] G. Davies, F. Kelly, “Network Dimensioning, Service Costing, and Pricing in a Packet-Switched Environment,” *Telecommunications Policy*, Vol. 28, No. 4, pp. 391–412.
- [**DEC 1990**] Digital Equipment Corporation, “In Memoriam: J. C. R. Licklider 1915–1990,” SRC Research Report 61, Aug. 1990. <http://www.memex.org/licklider.pdf>
- [**DeClercq 2002**] J. DeClercq, O. Paridaens, “Scalability Implications of Virtual Private Networks,” *IEEE Communications Magazine*, Vol. 40, No. 5 (May 2002), pp. 151–157.
- [**Demers 1990**] A. Demers, S. Keshav, S. Shenker, “Analysis and Simulation of a Fair Queueing Algorithm,” *Internetworking: Research and Experience*, Vol. 1, No. 1 (1990), pp. 3–26.
- [**Denning 1997**] D. Denning (Editor), P. Denning (Preface), *Internet Besieged: Countering Cyberspace Scofflaws*, Addison-Wesley, Reading, MA, 1997.
- [**dhc 2012**] IETF Dynamic Host Configuration working group homepage, <http://www.ietf.org/html.charters/dhc-charter.html>
- [**Dhungel 2012**] P. Dhungel, K. W. Ross, M. Steiner., Y. Tian, X. Hei, “Xunlei: Peer-Assisted Download Acceleration on a Massive Scale,” *Passive and Active Measurement Conference (PAM) 2012*, Vienna, 2012.

- [Diffie 1976]** W. Diffie, M. E. Hellman, “New Directions in Cryptography,” *IEEE Transactions on Information Theory*, Vol IT-22 (1976), pp. 644–654.
- [Diggavi 2004]** S. N. Diggavi, N. Al-Dahir, A. Stamoulis, R. Calderbank, “Great Expectations: The Value of Spatial Diversity in Wireless Networks,” *Proceedings of the IEEE*, Vol. 92, No. 2 (Feb. 2004).
- [Dilley 2002]** J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, B. Weihl, “Globally Distributed Content Delivery,” *IEEE Internet Computing* (Sept.–Oct. 2002).
- [Ding 2011]** Y. Ding, Y. Du, Y. Hu, Z. Liu, L. Wang, K. W. Ross, A. Ghose, “Broadcast Yourself: Understanding YouTube Uploaders,” *Proc. 2011 ACM Internet Measurement Conference* (Berlin).
- [Diot 2000]** C. Diot, B. N. Levine, B. Lyles, H. Kassem, D. Balensiefen, “Deployment Issues for the IP Multicast Service and Architecture,” *IEEE Network*, Vol. 14, No. 1 (Jan./Feb. 2000) pp. 78–88.
- [Dischinger 2007]** M. Dischinger, A. Haeberlen, K. Gummadi, S. Saroiu, “Characterizing residential broadband networks,” *Proc. 2007 ACM Internet Measurement Conference*, pp. 24–26.
- [Dimitropoulos 2007]** X. Dimitropoulos, D. Krioukov, M. Fomenkov, B. Huffaker, Y. Hyun, KC Claffy, G. Riley, “AS Relationships: Inference and Validation,” *ACM Computer Communication Review* (Jan. 2007).
- [DOCSIS 2004]** Data-over-cable service interface specifications: Radio-frequency interface specification. ITU-T J.112, 2004.
- [DOCSIS 2011]** Data-Over-Cable Service Interface Specifications, DOCSIS 3.0: MAC and Upper Layer Protocols Interface Specification, CM-SP-MULPIv3.0-I16-110623, 2011.
- [Dodge 2012]** M. Dodge, “An Atlas of Cyberspaces,” http://www.cybergeography.org/atlas_isp_maps.html
- [Donahoo 2001]** M. Donahoo, K. Calvert, *TCP/IP Sockets in C: Practical Guide for Programmers*, Morgan Kaufman, 2001.
- [Doucer 2002]** J. R. Doucer, “The Sybil Attack,” *First International Workshop on Peer-to-Peer Systems (IPTPS '02)* (Cambridge, MA, Mar. 2002).
- [DSL 2012]** DSL Forum homepage, <http://www.dslforum.org/>
- [Dhungel 2008]** P. Dhungel, D. Wu, B. Schonhorst, K.W. Ross, “A Measurement Study of Attacks on BitTorrent Leechers,” *7th International Workshop on Peer-to-Peer Systems (IPTPS 2008)* (Tampa Bay, FL, Feb. 2008).
- [Droms 2002]** R. Droms, T. Lemon, *The DHCP Handbook* (2nd Edition), SAMS Publishing, 2002.
- [Edney 2003]** J. Edney and W. A. Arbaugh, *Real 802.11 Security: Wi-Fi Protected Access and 802.11i*, Addison-Wesley Professional, 2003.
- [Edwards 2011]** W. K. Edwards, R. Grinter, R. Mahajan, D. Wetherall, “Advancing the State of Home Networking,” *Communications of the ACM*, Vol. 54, No. 6 (June 2011), pp. 62–71.
- [Eklund 2002]** K. Eklund, R. Marks, K. Stanswood, S. Wang, “IEEE Standard 802.16: A Technical Overview of the Wireless MAN Air Interface for Broadband Wireless Access,” *IEEE Communications Magazine* (June 2002), pp. 98–107.
- [Ellis 1987]** H. Ellis, “The Story of Non-Secret Encryption,” <http://jya.com/ellisdoc.htm>
- [Ericsson 2011]** Ericsson, “LTE—An Introduction,” www.ericsson.com/res/docs/2011/lte_an_introduction.pdf

- [Ericsson 2012]** Ericsson, “The Evolution of Edge,” http://www.ericsson.com/technology/whitepapers/broadband/evolution_of_EDGE.shtml
- [Estrin 1997]** D. Estrin, M. Handley, A. Helmy, P. Huang, D. Thaler, “A Dynamic Bootstrap Mechanism for Rendezvous-Based Multicast Routing,” *Proc. 1998 IEEE INFOCOM* (New York, NY, Apr. 1998).
- [Falkner 2007]** J. Falkner, M. Piatek, J.P. John, A. Krishnamurthy, T. Anderson, “Profiling a Million Sser DHT,” *Proc. 2007 ACM Internet Measurement Conference*.
- [Faloutsos 1999]** C. Faloutsos, M. Faloutsos, P. Faloutsos, “What Does the Internet Look Like? Empirical Laws of the Internet Topology,” *Proc. 1999 ACM SIGCOMM* (Boston, MA, Aug. 1999).
- [Farrington 2010]** N. Farrington, G. Porter, S. Radhakrishnan, H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, A. Vahdat, “Helios: A Hybrid Electrical/Optical Switch Architecture for Modular Data Centers,” *Proc. 2010 ACM SIGCOMM*.
- [Feamster 2004]** N. Feamster, J. Winick, J. Rexford, “A Model for BGP Routing for Network Engineering,” *Proc. 2004 ACM SIGMETRICS* (New York, NY, June 2004).
- [Feamster 2005]** N. Feamste, H. Balakrishnan, “Detecting BGP Configuration Faults with Static Analysis,” *NSDI* (May 2005).
- [Feldman 2005]** M. Feldman J. Chuang, “Overcoming Free-Riding Behavior in Peer-to-peer Systems,” *ACM SIGecom Exchanges* (July 2005).
- [Feldmeier 1995]** D. Feldmeier, “Fast Software Implementation of Error Detection Codes,” *IEEE/ACM Transactions on Networking*, Vol. 3, No. 6 (Dec. 1995), pp. 640–652.
- [FIPS 1995]** Federal Information Processing Standard, “Secure Hash Standard,” FIPS Publication 180-1. <http://www.itl.nist.gov/fipspubs/fip180-1.htm>
- [Floyd 1999]** S. Floyd, K. Fall, “Promoting the Use of End-to-End Congestion Control in the Internet,” *IEEE/ACM Transactions on Networking*, Vol. 6, No. 5 (Oct. 1998), pp. 458–472.
- [Floyd 2000]** S. Floyd, M. Handley, J. Padhye, J. Widmer, “Equation-Based Congestion Control for Unicast Applications,” *Proc. 2000 ACM SIGCOMM* (Stockholm, Sweden, Aug. 2000).
- [Floyd 2001]** S. Floyd, “A Report on Some Recent Developments in TCP Congestion Control,” *IEEE Communications Magazine* (Apr. 2001).
- [Floyd 2012]** S. Floyd, “References on RED (Random Early Detection) Queue Management,” <http://www.icir.org/floyd/red.html>
- [Floyd Synchronization 1994]** S. Floyd, V. Jacobson, “Synchronization of Periodic Routing Messages,” *IEEE/ACM Transactions on Networking*, Vol. 2, No. 2 (Apr. 1997) pp. 122–136.
- [Floyd TCP 1994]** S. Floyd, “TCP and Explicit Congestion Notification,” *ACM SIGCOMM Computer Communications Review*, Vol. 24, No. 5 (Oct. 1994), pp. 10–23.
- [Fluhrer 2001]** S. Fluhrer, I. Mantin, A. Shamir, “Weaknesses in the Key Scheduling Algorithm of RC4,” *Eighth Annual Workshop on Selected Areas in Cryptography*, (Toronto, Canada, Aug. 2002).
- [Fortz 2000]** B. Fortz, M. Thorup, “Internet Traffic Engineering by Optimizing OSPF Weights,” *Proc. 2000 IEEE INFOCOM* (Tel Aviv, Israel, Apr. 2000).
- [Fortz 2002]** B. Fortz, J. Rexford, M. Thorup, “Traffic Engineering with Traditional IP Routing Protocols,” *IEEE Communication Magazine* (Oct. 2002).

- [Fraleigh 2003]** C. Fraleigh, F. Tobagi, C. Diot, “Provisioning IP Backbone Networks to Support Latency Sensitive Traffic,” *Proc. 2003 IEEE INFOCOM* (San Francisco, CA, Mar. 2003).
- [Freedman 2004]** M. J. Freedman, E. Freudenthal, D. Mazires, “Democratizing Content Publication with Coral,” *USENIX NSDI*, 2004.
- [Friedman 1999]** T. Friedman, D. Towsley “Multicast Session Membership Size Estimation,” *Proc. 1999 IEEE INFOCOM* (New York, NY, Mar. 1999).
- [Frost 1994]** J. Frost, “BSD Sockets: A Quick and Dirty Primer,” <http://world.std.com/~jimf/papers/sockets/sockets.html>
- [FTTH Council 2011a]** FTTH Council, “NORTH AMERICAN FTTH STATUS—MARCH 31, 2011” (March 2011), www.ftthcouncil.org
- [FTTH Council 2011b]** FTTH Council, “2011 Broadband Consumer Research” (June 2011), www.ftthcouncil.org
- [Gallagher 1983]** R. G. Gallagher, P. A. Humblet, P. M. Spira, “A Distributed Algorithm for Minimum Weight-Spanning Trees,” *ACM Trans. on Programming Languages and Systems*, Vol. 1, No. 5 (Jan. 1983), pp. 66–77.
- [Gao 2001]** L. Gao, J. Rexford, “Stable Internet Routing Without Global Coordination,” *IEEE/ACM Transactions on Networking*, Vol. 9, No. 6 (Dec. 2001), pp. 681–692.
- [Garces-Erce 2003]** L. Garces-Erce, K. W. Ross, E. Biersack, P. Felber, G. Urvoy-Keller, “TOPLUS: Topology Centric Lookup Service,” *Fifth Int. Workshop on Networked Group Communications (NGC 2003)* (Munich, Sept. 2003) <http://cis.poly.edu/~ross/papers/TOPLUS.pdf>
- [Gartner 2003]** F. C. Gartner, “A Survey of Self-Stabilizing Spanning-Tree Construction Algorithms,” *Technical Report IC/2003/38*, Swiss Federal Institute of Technology (EPFL), School of Computer and Communication Sciences, June 10, 2003. http://ic2.epfl.ch/publications/documents/IC_TECH_REPORT_200338.pdf
- [Gauthier 1999]** L. Gauthier, C. Diot, and J. Kurose, “End-to-end Transmission Control Mechanisms for Multiparty Interactive Applications on the Internet,” *Proc. 1999 IEEE INFOCOM* (New York, NY, Apr. 1999).
- [Girard 1990]** A. Girard, *Routing and Dimensioning in Circuit-Switched Networks*, Addison-Wesley, Reading, MA, 1990.
- [Glitho 1998]** R. Glitho, “Contrasting OSI Systems Management to SNMP and TMN,” *Journal of Network and Systems Management*, Vol. 6, No. 2 (June 1998), pp. 113–131.
- [Gnutella 2009]** “The Gnutella Protocol Specification, v0.4” http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf
- [Goodman 1997]** David J. Goodman, *Wireless Personal Communications Systems*, Prentice-Hall, 1997.
- [Google Locations 2012]** Google data centers. <http://www.google.com/corporate/datacenter/locations.html>
- [Goralski 1999]** W. Goralski, *Frame Relay for High-Speed Networks*, John Wiley, New York, 1999.
- [Goralski 2001]** W. Goralski, *Optical Networking and WDM*, Osborne/McGraw-Hill, Berkeley, CA, 2001.
- [Greenberg 2009a]** A. Greenberg, J. Hamilton, D. Maltz, P. Patel, “The Cost of a Cloud: Research Problems in Data Center Networks,” *ACM Computer Communications Review* (Jan. 2009).

- [Greenberg 2009b] A. Greenberg, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, S. Sengupta, “VL2: A Scalable and Flexible Data Center Network,” *Proc. 2009 ACM SIGCOMM*.
- [Greenberg 2011] A. Greenberg, J. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, S. Sengupta, “VL2: A Scalable and Flexible Data Center Network,” *Communications of the ACM*, Vol. 54, No. 3 (Mar. 2011), pp. 95–104.
- [Griffin 2012] T. Griffin, “Interdomain Routing Links,” <http://www.cl.cam.ac.uk/~tgg22/interdomain/>
- [Guha 2006] S. Guha, N. Daswani, R. Jain, “An Experimental Study of the Skype Peer-to-Peer VoIP System,” *Proc. Fifth Int. Workshop on P2P Systems* (Santa Barbara, CA, 2006).
- [Guo 2005] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, X. Zhang, “Measurement, Analysis, and Modeling of BitTorrent-Like Systems,” *Proc. 2005 ACM Internet Measurement Conference*.
- [Guo 2009] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, S. Lu, “BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers,” *Proc. 2009 ACM SIGCOMM*.
- [Gupta 2001] P. Gupta, N. McKeown, “Algorithms for Packet Classification,” *IEEE Network Magazine*, Vol. 15, No. 2 (Mar./Apr. 2001), pp. 24–32.
- [Ha 2008] Ha, S., Rhee, I., L. Xu, “CUBIC: A New TCP-Friendly High-Speed TCP Variant,” *ACM SIGOPS Operating System Review*, 2008.
- [Halabi 2000] S. Halabi, *Internet Routing Architectures*, 2nd Ed., Cisco Press, 2000.
- [Halperin 2008] D. Halperin, T. Heydt-Benjamin, B. Ransford, S. Clark, B. Defend, W. Morgan, K. Fu, T. Kohno, W. Maisel, “Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses,” *Proc. 29th Annual IEEE Symposium on Security and Privacy* (May 2008).
- [Halperin 2011] D. Halperin, S. Kandula, J. Padhye, P. Bahl, D. Wetherall, “Augmenting Data Center Networks with Multi-Gigabit Wireless Links,” *Proc. 2011 ACM SIGCOMM*.
- [Hanabali 2005] A. A. Hanbali, E. Altman, P. Nain, “A Survey of TCP over Ad Hoc Networks,” *IEEE Commun. Surveys and Tutorials*, Vol. 7, No. 3 (2005), pp. 22–36.
- [Hei 2007] X. Hei, C. Liang, J. Liang, Y. Liu, K. W. Ross, “A Measurement Study of a Large-scale P2P IPTV System,” *IEEE Trans. on Multimedia* (Dec. 2007).
- [Heidemann 1997] J. Heidemann, K. Obraczka, J. Touch, “Modeling the Performance of HTTP over Several Transport Protocols,” *IEEE/ACM Transactions on Networking*, Vol. 5, No. 5 (Oct. 1997), pp. 616–630.
- [Held 2001] G. Held, *Data Over Wireless Networks: Bluetooth, WAP, and Wireless LANs*, McGraw-Hill, 2001.
- [Hersent 2000] O. Hersent, D. Gurle, J.-P. Petit, *IP Telephony: Packet-Based Multimedia Communication Systems*, Pearson Education Limited, Edinburgh, 2000.
- [Holland 2001] G. Holland, N. Vaidya, V. Bahl, “A Rate-Adaptive MAC Protocol for Multi-Hop Wireless Networks,” *Proc. 2001 ACM Int. Conference of Mobile Computing and Networking (Mobicom01)* (Rome, Italy, July 2001).
- [Hollot 2002] C.V. Hollot, V. Misra, D. Towsley, W. Gong, “Analysis and design of controllers for AQM routers supporting TCP flows,” *IEEE Transactions on Automatic Control*, Vol. 47, No. 6 (June 2002), pp. 945–959.

- [**Huang 2002**] C. Haung, V. Sharma, K. Owens, V. Makam, “Building Reliable MPLS Networks Using a Path Protection Mechanism,” *IEEE Communications Magazine*, Vol. 40, No. 3 (Mar. 2002), pp. 156–162.
- [**Huang 2005**] Y. Huang, R. Guerin, “Does Over-Provisioning Become More or Less Efficient as Networks Grow Larger?,” *Proc. IEEE Int. Conf. Network Protocols (ICNP)* (Boston MA, November 2005).
- [**Huang 2007**] C. Huang, Jin Li, K.W. Ross, “Can Internet VoD Be Profitable?,” *Proc 2007 ACM SIGCOMM* (Kyoto, Aug. 2007).
- [**Huang 2008**] C. Huang, J. Li, A. Wang, K. W. Ross, “Understanding Hybrid CDN-P2P: Why Limelight Needs its Own Red Swoosh,” *Proc. 2008 NOSSDAV*, Braunschweig, Germany.
- [**Huang 2010**] C. Huang, N. Holt, Y. A. Wang, A. Greenberg, J. Li, K. W. Ross, “A DNS Reflection Method for Global Traffic Management,” *Proc. 2010 USENIX*, Boston.
- [**Huitema 1998**] C. Huitema, *IPv6: The New Internet Protocol*, 2nd Ed., Prentice Hall, Englewood Cliffs, NJ, 1998.
- [**Huston 1999a**] G. Huston, “Interconnection, Peering, and Settlements—Part I,” *The Internet Protocol Journal*, Vol. 2, No. 1 (Mar. 1999).
- [**Huston 2004**] G. Huston, “NAT Anatomy: A Look Inside Network Address Translators,” *The Internet Protocol Journal*, Vol. 7, No. 3 (Sept. 2004).
- [**Huston 2008a**] G. Huston, “Confronting IPv4 Address Exhaustion,” <http://www.potaroo.net/ispcol/2008-10/v4depletion.html>
- [**Huston 2008b**] G. Huston, G. Michaelson, “IPv6 Deployment: Just where are we?” <http://www.potaroo.net/ispcol/2008-04/ipv6.html>
- [**Huston 2011a**] G. Huston, “A Rough Guide to Address Exhaustion,” *The Internet Protocol Journal*, Vol. 14, No. 1 (Mar. 2011).
- [**Huston 2011b**] G. Huston, “Transitioning Protocols,” *The Internet Protocol Journal*, Vol. 14, No. 1 (Mar. 2011).
- [**IAB 2012**] Internet Architecture Board homepage, <http://www.iab.org/>
- [**IANA 2012a**] Internet Assigned Number Authority homepage, <http://www.iana.org/>
- [**IANA 2012b**] Internet Assigned Number Authority, “Private Enterprise Numbers” <http://www.iana.org/assignments/enterprise-numbers>
- [**IANA Protocol Numbers 2012**] Internet Assigned Numbers Authority, Protocol Numbers, <http://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml>
- [**IANA TLD 2012**] IANA Root Zone Database, <http://www.iana.org/domains/root/db/>
- [**ICANN 2012**] The Internet Corporation for Assigned Names and Numbers homepage, <http://www.icann.org>
- [**IEC Optical 2012**] IEC Online Education, “Optical Access,” http://www.iec.org/online/tutorials/opt_acc/
- [**IEEE 802 2012**] IEEE 802 LAN/MAN Standards Committee homepage, <http://www.ieee802.org/>
- [**IEEE 802.11 1999**] IEEE 802.11, “1999 Edition (ISO/IEC 8802-11: 1999) IEEE Standards for Information Technology—Telecommunications and Information Exchange Between Systems—Local and Metropolitan Area Network—Specific Requirements—Part 11:

- Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification,” <http://standards.ieee.org/getieee802/download/802.11-1999.pdf>
- [IEEE 802.11n 2012] IEEE, “IEEE P802.11—Task Group N—Meeting Update: Status of 802.11n,” http://grouper.ieee.org/groups/802/11/Reports/tgn_update.htm
- [IEEE 802.15 2012] IEEE 802.15 Working Group for WPAN homepage, <http://grouper.ieee.org/groups/802/15/>.
- [IEEE 802.15.4 2012] IEEE 802.15 WPAN Task Group 4, <http://www.ieee802.org/15/pub/TG4.html>
- [IEEE 802.16d 2004] IEEE, “IEEE Standard for Local and Metropolitan Area Networks, Part 16: Air Interface for Fixed Broadband Wireless Access Systems,” <http://standards.ieee.org/getieee802/download/802.16-2004.pdf>
- [IEEE 802.16e 2005] IEEE, “IEEE Standard for Local and Metropolitan Area Networks, Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems, Amendment 2: Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licensed Bands and Corrigendum 1,” <http://standards.ieee.org/getieee802/download/802.16e-2005.pdf>
- [IEEE 802.1q 2005] IEEE, “IEEE Standard for Local and Metropolitan Area Networks: Virtual Bridged Local Area Networks,” <http://standards.ieee.org/getieee802/download/802.1Q-2005.pdf>
- [IEEE 802.1X] IEEE Std 802.1X-2001 Port-Based Network Access Control, http://standards.ieee.org/reading/ieee/std_public/description/lanman/802.1x-2001_desc.html
- [IEEE 802.3 2012] IEEE, “IEEE 802.3 CSMA/CD (Ethernet),” <http://grouper.ieee.org/groups/802/3/>
- [IEEE 802.5 2012] IEEE, IEEE 802.5 homepage, <http://www.ieee802.org/5/www8025org/>
- [IETF 2012] Internet Engineering Task Force homepage, <http://www.ietf.org>
- [Ihm 2011] S. Ihm, V. S. Pai, “Towards Understanding Modern Web Traffic,” *Proc. 2011 ACM Internet Measurement Conference* (Berlin).
- [IMAP 2012] The IMAP Connection, <http://www imap.org/>
- [Intel 2012] Intel Corp, “Intel® 82544 Gigabit Ethernet Controller,” http://www.intel.com/design/network/products/lan/docs/82544_docs.htm
- [Intel WiMax 2012] Intel Corp., “WiMax Technology,” <http://www.intel.com/technology/wimax/index.htm>
- [Internet2 Multicast 2012] Internet2 Multicast Working Group homepage, <http://www.internet2.edu/multicast/>
- [IPv6 2012] IPv6.com homepage, <http://www.ipv6.com/>
- [ISC 2012] Internet Systems Consortium homepage, <http://www.isc.org>
- [ISI 1979] Information Sciences Institute, “DoD Standard Internet Protocol,” Internet Engineering Note 123 (Dec. 1979), <http://www.isi.edu/in-notes/ien/ien123.txt>
- [ISO 2012] International Organization for Standardization homepage, International Organization for Standardization, <http://www.iso.org/>
- [ISO X.680 2002] International Organization for Standardization, “X.680: ITU-T Recommendation X.680 (2002) Information Technology—Abstract Syntax Notation One (ASN.1): Specification of Basic Notation.” <http://www.itu.int/ITU-T/studygroups/com17/languages/X.680-0207.pdf>

- [ITU 1999] Asymmetric Digital Subscriber Line (ADSL) Transceivers. ITU-T G.992.1, 1999.
- [ITU 2003] Asymmetric Digital Subscriber Line (ADSL) Transceivers—Extended Bandwidth ADSL2 (ADSL2Plus). ITU-T G.992.5, 2003.
- [ITU 2005a] International Telecommunication Union, “ITU-T X.509, The Directory: Public-key and attribute certificate frameworks” (August 2005).
- [ITU 2005b] International Telecommunication Union, *The Internet of Things*, 2005, http://www.itu.int/osg/spu/publications/internetofthings/InternetofThings_summary.pdf
- [ITU 2012] The ITU homepage, <http://www.itu.int/>
- [ITU Statistics 2012] International Telecommunications Union, “ICT Statistics,” <http://www.itu.int/ITU-D/icteye/Reports.aspx>
- [ITU 2011] ITU, “Measuring the Information Society, 2011,” <http://www.itu.int/ITU-D/ict/publications/idi/2011/index.html>
- [ITU 2011] ITU, “The World in 2010: ICT Facts and Figures,” http://www.itu.int/ITU-D/ict/material/Telecom09_flyer.pdf
- [ITU-T Q.2931 1995] International Telecommunication Union, “Recommendation Q.2931 (02/95) - Broadband Integrated Services Digital Network (B-ISDN)—Digital subscriber signalling system no. 2 (DSS 2)—User-network interface (UNI)—Layer 3 specification for basic call/connection control.”
- [Iyer 2002] S. Iyer, R. Zhang, N. McKeown, “Routers with a Single Stage of Buffering,” *Proc. 2002 ACM SIGCOMM* (Pittsburgh, PA, Aug. 2002).
- [Iyer 2008] S. Iyer, R. R. Kompella, N. McKeown, “Designing Packet Buffers for Router Line Cards,” *IEEE Transactions on Networking*, Vol. 16, No. 3 (June 2008), pp. 705–717.
- [Jacobson 1988] V. Jacobson, “Congestion Avoidance and Control,” *Proc. 1988 ACM SIGCOMM* (Stanford, CA, Aug. 1988), pp. 314–329.
- [Jain 1986] R. Jain, “A timeout-based congestion control scheme for window flow-controlled networks,” *IEEE Journal on Selected Areas in Communications SAC-4*, 7 (Oct. 1986).
- [Jain 1989] R. Jain, “A Delay-Based Approach for Congestion Avoidance in Interconnected Heterogeneous Computer Networks,” *ACM SIGCOMM Computer Communications Review*, Vol. 19, No. 5 (1989), pp. 56–71.
- [Jain 1994] R. Jain, *FDDI Handbook: High-Speed Networking Using Fiber and Other Media*, Addison-Wesley, Reading, MA, 1994.
- [Jain 1996] R. Jain, S. Kalyanaraman, S. Fahmy, R. Goyal, S. Kim, “Tutorial Paper on ABR Source Behavior,” *ATM Forum/96-1270*, Oct. 1996. <http://www.cse.wustl.edu/~jain/atmf/ftp/atm96-1270.pdf>
- [Jaiswal 2003] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, D. Towsley, “Measurement and Classification of Out-of-Sequence Packets in a Tier-1 IP backbone,” *Proc. 2003 IEEE INFOCOM*.
- [Ji 2003] P. Ji, Z. Ge, J. Kurose, D. Towsley, “A Comparison of Hard-State and Soft-State Signaling Protocols,” *Proc. 2003 ACM SIGCOMM* (Karlsruhe, Germany, Aug. 2003).
- [Jiang 2001] W. Jiang, J. Lennox, H. Schulzrinne, K. Singh, “Towards Junking the PBX: Deploying IP Telephony,” *NOSSDAV'01* (Port Jefferson, NY, June 2001).
- [Jimenez 1997] D. Jimenez, “Outside Hackers Infiltrate MIT Network, Compromise Security,” *The Tech*, Vol. 117, No 49 (Oct. 1997), p. 1, <http://www-tech.mit.edu/V117/N49/hackers.49n.html>
- [Jin 2004] C. Jin, D. X. We, S. Low, “FAST TCP: Motivation, architecture, algorithms, performance,” *Proc. 2004 IEEE INFOCOM* (Hong Kong, March 2004).

- [**Kaarinen 2001**] H. Kaaranen, S. Naghian, L. Laitinen, A. Ahtiainen, V. Niemi, *Networks: Architecture, Mobility and Services*, New York: John Wiley & Sons, 2001.
- [**Kahn 1967**] D. Kahn, *The Codebreakers: The Story of Secret Writing*, The Macmillan Company, 1967.
- [**Kahn 1978**] R. E. Kahn, S. Gronemeyer, J. Burchfiel, R. Kunzelman, “Advances in Packet Radio Technology,” *Proc. 1978 IEEE INFOCOM*, 66, 11 (Nov. 1978).
- [**Kamerman 1997**] A. Kamerman, L. Monteban, “WaveLAN-II: A High-Performance Wireless LAN for the Unlicensed Band,” *Bell Labs Technical Journal* (Summer 1997), pp. 118–133.
- [**Kangasharju 2000**] J. Kangasharju, K. W. Ross, J. W. Roberts, “Performance Evaluation of Redirection Schemes in Content Distribution Networks,” *Proc. 5th Web Caching and Content Distribution Workshop* (Lisbon, Portugal, May 2000).
- [**Kar 2000**] K. Kar, M. Kodialam, T. V. Lakshman, “Minimum Interference Routing of Bandwidth Guaranteed Tunnels with MPLS Traffic Engineering Applications,” *IEEE J. Selected Areas in Communications* (Dec. 2000).
- [**Karn 1987**] P. Karn, C. Partridge, “Improving Round-Trip Time Estimates in Reliable Transport Protocols,” *Proc. 1987 ACM SIGCOMM*.
- [**Karol 1987**] M. Karol, M. Hluchyj, A. Morgan, “Input Versus Output Queuing on a Space-Division Packet Switch,” *IEEE Transactions on Communications*, Vol. 35, No. 12 (Dec. 1987), pp. 1347–1356.
- [**Katabi 2002**] D. Katabi, M. Handley, C. Rohrs, “Internet Congestion Control for Future High Bandwidth-Delay Product Environments,” *Proc. 2002 ACM SIGCOMM* (Pittsburgh, PA, Aug. 2002).
- [**Katzela 1995**] I. Katzela, M. Schwartz. “Schemes for Fault Identification in Communication Networks,” *IEEE/ACM Transactions on Networking*, Vol. 3, No. 6 (Dec. 1995), pp. 753–764.
- [**Kaufman 1995**] C. Kaufman, R. Perlman, M. Speciner, *Network Security, Private Communication in a Public World*, Prentice Hall, Englewood Cliffs, NJ, 1995.
- [**Kelly 1998**] F. P. Kelly, A. Maulloo, D. Tan, “Rate control for communication networks: Shadow prices, proportional fairness and stability,” *J. Operations Res. Soc.*, Vol. 49, No. 3 (Mar. 1998), pp. 237–252.
- [**Kelly 2003**] T. Kelly, “Scalable TCP: improving performance in high speed wide area networks,” *ACM SIGCOMM Computer Communications Review*, Volume 33, No. 2 (Apr. 2003), pp. 83–91.
- [**Kilkki 1999**] K. Kilkki, *Differentiated Services for the Internet*, Macmillan Technical Publishing, Indianapolis, IN, 1999.
- [**Kim 2005**] H. Kim, S. Rixner, V. Pai, “Network Interface Data Caching,” *IEEE Transactions on Computers*, Vol. 54, No. 11 (Nov. 2005), pp. 1394–1408.
- [**Kim 2008**] C. Kim, M. Caesar, J. Rexford, “Floodless in SEATTLE: A Scalable Ethernet Architecture for Large Enterprises,” *Proc. 2008 ACM SIGCOMM* (Seattle, WA, Aug. 2008).
- [**Kleinrock 1961**] L. Kleinrock, “Information Flow in Large Communication Networks,” RLE Quarterly Progress Report, July 1961.
- [**Kleinrock 1964**] L. Kleinrock, *1964 Communication Nets: Stochastic Message Flow and Delay*, McGraw-Hill, New York, NY, 1964.
- [**Kleinrock 1975**] L. Kleinrock, *Queueing Systems, Vol. 1*, John Wiley, New York, 1975.

- [Kleinrock 1975b]** L. Kleinrock, F. A. Tobagi, “Packet Switching in Radio Channels: Part I—Carrier Sense Multiple-Access Modes and Their Throughput-Delay Characteristics,” *IEEE Transactions on Communications*, Vol. 23, No. 12 (Dec. 1975), pp. 1400–1416.
- [Kleinrock 1976]** L. Kleinrock, *Queueing Systems, Vol. 2*, John Wiley, New York, 1976.
- [Kleinrock 2004]** L. Kleinrock, “The Birth of the Internet,” <http://www.lk.cs.ucla.edu/LK/Inet/birth.html>
- [Kohler 2006]** E. Kohler, M. Handley, S. Floyd, “DDCP: Designing DCCP: Congestion Control Without Reliability,” *Proc. 2006 ACM SIGCOMM* (Pisa, Italy, Sept. 2006).
- [Kolding 2003]** T. Kolding, K. Pedersen, J. Wigard, F. Frederiksen, P. Mogensen, “High Speed Downlink Packet Access: WCDMA Evolution,” *IEEE Vehicular Technology Society News* (Feb. 2003), pp. 4–10.
- [Koponen 2011]** T. Koponen, S. Shenker, H. Balakrishnan, N. Feamster, I. Ganichev, A. Ghodsi, P. B. Godfrey, N. McKeown, G. Parulkar, B. Raghavan, J. Rexford, S. Arianfar, D. Kuptsov, “Architecting for Innovation,” *ACM Computer Communications Review*, 2011.
- [Korhonen 2003]** J. Korhonen, *Introduction to 3G Mobile Communications*, 2nd ed., Artech House, 2003.
- [Koziol 2003]** J. Koziol, *Intrusion Detection with Snort*, Sams Publishing, 2003.
- [Krishnamurthy 2001]** B. Krishnamurthy, and J. Rexford, *Web Protocols and Practice: HTTP/1.1, Networking Protocols, and Traffic Measurement*, Addison-Wesley, Boston, MA, 2001.
- [Krishnamurthy 2001b]** B. Krishnamurthy, C. Wills, Y. Zhang, “On the Use and Performance of Content Distribution Networks,” *Proc. 2001 ACM Internet Measurement Conference*.
- [Krishnan 2009]** R. Krishnan, H. Madhyastha, S. Srinivasan, S. Jain, A. Krishnamurthy, T. Anderson, J. Gao, “Moving Beyond End-to-end Path Information to Optimize CDN Performance,” *Proc. 2009 ACM Internet Measurement Conference*.
- [Kulkarni 2005]** S. Kulkarni, C. Rosenberg, “Opportunistic Scheduling: Generalizations to Include Multiple Constraints, Multiple Interfaces, and Short Term Fairness,” *Wireless Networks*, 11 (2005), 557–569.
- [Kumar 2006]** R. Kumar, K.W. Ross, “Optimal Peer-Assisted File Distribution: Single and Multi-Class Problems,” *IEEE Workshop on Hot Topics in Web Systems and Technologies* (Boston, MA, 2006).
- [Labovitz 1997]** C. Labovitz, G. R. Malan, F. Jahanian, “Internet Routing Instability,” *Proc. 1997 ACM SIGCOMM* (Cannes, France, Sept. 1997), pp. 115–126.
- [Labovitz 2010]** C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, F. Jahanian, “Internet Inter-Domain Traffic,” *Proc. 2010 ACM SIGCOMM*.
- [Labrador 1999]** M. Labrador, S. Banerjee, “Packet Dropping Policies for ATM and IP Networks,” *IEEE Communications Surveys*, Vol. 2, No. 3 (Third Quarter 1999), pp. 2–14.
- [Lacage 2004]** M. Lacage, M.H. Manshaei, T. Turletti, “IEEE 802.11 Rate Adaptation: A Practical Approach,” *ACM Int. Symposium on Modeling, Analysis, and Simulation of Wireless and Mobile Systems (MSWiM)* (Venice, Italy, Oct. 2004).
- [Lakhina 2004]** A. Lakhina, M. Crovella, C. Diot, “Diagnosing Network-Wide Traffic Anomalies,” *Proc. 2004 ACM SIGCOMM*.
- [Lakhina 2005]** A. Lakhina, M. Crovella, C. Diot, “Mining Anomalies Using Traffic Feature Distributions,” *Proc. 2005 ACM SIGCOMM*.

- [Lakshman 1997]** T. V. Lakshman, U. Madhow, “The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss,” *IEEE/ACM Transactions on Networking*, Vol. 5, No. 3 (1997), pp. 336–350.
- [Lam 1980]** S. Lam, “A Carrier Sense Multiple Access Protocol for Local Networks,” *Computer Networks*, Vol. 4 (1980), pp. 21–32.
- [Larmouth 1996]** J. Larmouth, *Understanding OSI*, International Thomson Computer Press 1996. Chapter 8 of this book deals with ASN.1 and is available online at <http://www.salford.ac.uk/iti/books/osi/all.html#head8>.
- [Larmouth 2012]** J. Larmouth, *Understanding OSI*, <http://www.business.salford.ac.uk/legacy/isi/books/osi/osi.html>
- [Lawton 2001]** G. Lawton, “Is IPv6 Finally Gaining Ground?” *IEEE Computer Magazine* (Aug. 2001), pp. 11–15.
- [LeBlond 2011]** S. LeBlond, C. Zhang, A. Legout, K. W. Ross, W. Dabbous, “Exploring the Privacy Limits of Real-Time Communication Applications,” *Proc. 2011 ACM Internet Measurement Conference* (Berlin, 2011).
- [LeBlond 2011]** S. LeBlond, C. Zhang, A. Legout, K. W. Ross, W. Dabbous, “I Know Where You and What You Are Sharing: Exploiting P2P Communications to Invade Users Privacy,” *Proc. 2011 ACM Internet Measurement Conference* (Berlin).
- [Leighton 2009]** T. Leighton, “Improving Performance on the Internet,” *Communications of the ACM*, Vol. 52, No. 2 (Feb. 2009), pp. 44–51.
- [Leiner 1998]** B. Leiner, V. Cerf, D. Clark, R. Kahn, L. Kleinrock, D. Lynch, J. Postel, L. Roberts, S. Woolf, “A Brief History of the Internet,” <http://www.isoc.org/internet/history/brief.html>
- [Leung 2006]** K. Leung, V. O.K. Li, “TCP in Wireless Networks: Issues, Approaches, and Challenges,” *IEEE Commun. Surveys and Tutorials*, Vol. 8, No. 4 (2006), pp. 64–79.
- [Li 2004]** L. Li, D. Alderson, W. Willinger, J. Doyle, “A First-Principles Approach to Understanding the Internet’s Router-Level Topology,” *Proc. 2004 ACM SIGCOMM* (Portland, OR, Aug. 2004).
- [Li 2007]** J. Li, M. Guidero, Z. Wu, E. Purpus, T. Ehrenkranz, “BGP Routing Dynamics Revisited.” *ACM Computer Communication Review* (April 2007).
- [Liang 2006]** J. Liang, N. Naoumov, K.W. Ross, “The Index Poisoning Attack in P2P File-Sharing Systems,” *Proc. 2006 IEEE INFOCOM* (Barcelona, Spain, April 2006).
- [Lin 2001]** Y. Lin, I. Chlamtac, *Wireless and Mobile Network Architectures*, John Wiley and Sons, New York, NY, 2001.
- [Liogkas 2006]** N. Liogkas, R. Nelson, E. Kohler, L. Zhang, “Exploiting BitTorrent For Fun (But Not Profit),” *6th International Workshop on Peer-to-Peer Systems (IPTPS 2006)*.
- [Liu 2002]** B. Liu, D. Goeckel, D. Towsley, “TCP-Cognizant Adaptive Forward Error Correction in Wireless Networks,” *Proc. 2002 Global Internet*.
- [Liu 2003]** J. Liu, I. Matta, M. Crovella, “End-to-End Inference of Loss Nature in a Hybrid Wired/Wireless Environment,” *Proc. WiOpt’03: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*.
- [Liu 2010]** Z. Liu, P. Dhungel, Di Wu, C. Zhang, K. W. Ross, “Understanding and Improving Incentives in Private P2P Communities,” *ICDCS* (Genoa, Italy, 2010).

- [Locher 2006]** T. Locher, P. Moor, S. Schmid, R. Wattenhofer, “Free Riding in BitTorrent is Cheap,” *Proc. ACM HotNets 2006* (Irvine CA, Nov. 2006).
- [Lui 2004]** J. Lui, V. Misra, D. Rubenstein, “On the Robustness of Soft State Protocols,” *Proc. IEEE Int. Conference on Network Protocols (ICNP '04)*, pp. 50–60.
- [Luotonen 1998]** A. Luotonen, *Web Proxy Servers*, Prentice Hall, Englewood Cliffs, NJ, 1998.
- [Lynch 1993]** D. Lynch, M. Rose, *Internet System Handbook*, Addison-Wesley, Reading, MA, 1993.
- [Macedonia 1994]** M. Macedonia, D. Brutzman, “MBone Provides Audio and Video Across the Internet,” *IEEE Computer Magazine*, Vol. 27, No. 4 (Apr. 1994), pp. 30–36.
- [Mahdavi 1997]** J. Mahdavi, S. Floyd, “TCP-Friendly Unicast Rate-Based Flow Control,” unpublished note (Jan. 1997).
- [Malware 2006]** Computer Economics, “2005 Malware Report: The Impact of Malicious Code Attacks,” <http://www.computereconomics.com>
- [manet 2012]** IETF Mobile Ad-hoc Networks (manet) Working Group, <http://www.ietf.org/html.charters/manet-charter.html>
- [Mao 2002]** Z. M. Mao, C. Cranor, F. Boudlis, M. Rabinovich, O. Spatscheck, J. Wang, “A Precise and Efficient Evaluation of the Proximity Between Web Clients and Their Local DNS Servers,” *Proc. 2002 USENIX ATC*.
- [MaxMind 2012]** <http://www.maxmind.com/app/ip-location>
- [Maymounkov 2002]** P. Maymounkov, D. Mazières, “Kademlia: A Peer-to-Peer Information System Based on the XOR Metric.” *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)* (Mar. 2002), pp. 53–65.
- [McKeown 1997a]** N. McKeown, M. Izzard, A. Mekkittikul, W. Ellersick, M. Horowitz, “The Tiny Tera: A Packet Switch Core,” *IEEE Micro Magazine* (Jan.–Feb. 1997).
- [McKeown 1997b]** N. McKeown, “A Fast Switched Backplane for a Gigabit Switched Router,” *Business Communications Review*, Vol. 27, No. 12. http://tiny-tera.stanford.edu/~nickm/papers/cisco_fasts_wp.pdf
- [McKeown 2008]** N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, “OpenFlow: Enabling Innovation in Campus Networks,” *ACM SIGCOMM Computer Communication Review*, Vol. 38, No. 2 (Apr. 2008).
- [McQuillan 1980]** J. McQuillan, I. Richer, E. Rosen, “The New Routing Algorithm for the Arpanet,” *IEEE Transactions on Communications*, Vol. 28, No. 5 (May 1980), pp. 711–719.
- [Medhi 1997]** D. Medhi, D. Tipper (eds.), Special Issue: Fault Management in Communication Networks, *Journal of Network and Systems Management*, Vol. 5, No. 2 (June 1997).
- [Metcalfe 1976]** R. M. Metcalfe, D. R. Boggs, “Ethernet: Distributed Packet Switching for Local Computer Networks,” *Communications of the Association for Computing Machinery*, Vol. 19, No. 7 (July 1976), pp. 395–404.
- [Meyers 2004]** A. Myers, T. Ng, H. Zhang, “Rethinking the Service Model: Scaling Ethernet to a Million Nodes,” *ACM Hotnets Conference*, 2004.
- [MFA Forum 2012]** IP/MPLS Forum homepage, <http://www.ipmplsforum.org/>
- [Mirkovic 2005]** J. Mirkovic, S. Dietrich, D. Dittrich, P. Reiher, *Internet Denial of Service: Attack and Defense Mechanisms*, Prentice Hall, 2005.
- [Mockapetris 1988]** P. V. Mockapetris, K. J. Dunlap, “Development of the Domain Name System,” *Proc. 1988 ACM SIGCOMM* (Stanford, CA, Aug. 1988).

- [Mockapetris 2005] P. Mockapetris, Sigcomm Award Lecture, video available at <http://www.postel.org/sigcomm>
- [Mogul 2003] J. Mogul, “TCP offload is a dumb idea whose time has come,” *Proc. HotOS IX: The 9th Workshop on Hot Topics in Operating Systems* (2003), USENIX Association.
- [Molinero-Fernandez 2002] P. Molinaro-Fernandez, N. McKeown, H. Zhang, “Is IP Going to Take Over the World (of Communications)?,” *Proc. 2002 ACM Hotnets*.
- [Molle 1987] M. L. Molle, K. Sohraby, A. N. Venetsanopoulos, “Space-Time Models of Asynchronous CSMA Protocols for Local Area Networks,” *IEEE Journal on Selected Areas in Communications*, Vol. 5, No. 6 (1987), pp. 956–968.
- [Moore 2001] D. Moore, G. Voelker, S. Savage, “Inferring Internet Denial of Service Activity,” *Proc. 2001 USENIX Security Symposium* (Washington, DC, Aug. 2001).
- [Moore 2003] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, N. Weaver, “Inside the Slammer Worm,” *2003 IEEE Security and Privacy Conference*.
- [Moshchuk 2006] A. Moshchuk, T. Bragin, S. Gribble, H. Levy, “A Crawler-based Study of Spyware on the Web,” *Proc. 13th Annual Network and Distributed Systems Security Symposium (NDSS 2006)* (San Diego, CA, Feb. 2006).
- [Motorola 2007] Motorola, “Long Term Evolution (LTE): A Technical Overview,” http://www.motorola.com/staticfiles/Business/Solutions/Industry%20Solutions/Service%20Providers/Wireless%20Operators/LTE/_Document/Static%20Files/6834_MotDoc_New.pdf
- [Mouly 1992] M. Mouly, M. Pautet, *The GSM System for Mobile Communications*, Cell and Sys, Palaiseau, France, 1992.
- [Moy 1998] J. Moy, *OSPF: Anatomy of An Internet Routing Protocol*, Addison-Wesley, Reading, MA, 1998.
- [Mudigonda 2011] J. Mudigonda, P. Yalagandula, J. C. Mogul, B. Stiekes, Y. Pouffary, “NetLord: A Scalable Multi-Tenant Network Architecture for Virtualized Datacenters,” *Proc. 2011 ACM SIGCOMM*.
- [Mukherjee 1997] B. Mukherjee, *Optical Communication Networks*, McGraw-Hill, 1997.
- [Mukherjee 2006] B. Mukherjee, *Optical WDM Networks*, Springer, 2006.
- [Mydotr 2009] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, A. Vahdat, “PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric,” *Proc. 2009 ACM SIGCOMM*.
- [Nadel 2011] B. Nadel, “4G shootout: Verizon LTE vs. Sprint WiMax,” *Computerworld*, February 3, 2011.
- [Nahum 2002] E. Nahum, T. Barzilai, D. Kandlur, “Performance Issues in WWW Servers,” *IEEE/ACM Transactions on Networking*, Vol 10, No. 1 (Feb. 2002).
- [Naoumov 2006] N. Naoumov, K.W. Ross, “Exploiting P2P Systems for DDoS Attacks,” *Intl Workshop on Peer-to-Peer Information Management* (Hong Kong, May 2006),
- [Neglia 2007] G. Neglia, G. Reina, H. Zhang, D. Towsley, A. Venkataramani, J. Danaher, “Availability in BitTorrent Systems,” *Proc. 2007 IEEE INFOCOM*.
- [Neumann 1997] R. Neumann, “Internet Routing Black Hole,” *The Risks Digest: Forum on Risks to the Public in Computers and Related Systems*, Vol. 19, No. 12 (May 1997). <http://catless.ncl.ac.uk/Risks/19.12.html#subj1.1>
- [Neville-Neil 2009] G. Neville-Neil, “Whither Sockets?” *Communications of the ACM*, Vol. 52, No. 6 (June 2009), pp. 51–55.

- [Nicholson 2006]** A Nicholson, Y. Chawathe, M. Chen, B. Noble, D. Wetherall, “Improved Access Point Selection,” *Proc. 2006 ACM Mobicom Conference* (Uppsala Sweden, 2006).
- [Nielsen 1997]** H. F. Nielsen, J. Gettys, A. Baird-Smith, E. Prud’hommeaux, H. W. Lie, C. Lilley, “Network Performance Effects of HTTP/1.1, CSS1, and PNG,” *W3C Document*, 1997 (also appears in *Proc. 1997 ACM SIGCOMM* (Cannes, France, Sept 1997), pp. 155–166.
- [NIST 2001]** National Institute of Standards and Technology, “Advanced Encryption Standard (AES),” Federal Information Processing Standards 197, Nov. 2001, <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>
- [NIST IPv6 2012]** National Institute of Standards, “Estimating IPv6 & DNSSEC Deployment SnapShots,” <http://usgv6-deploymon.antd.nist.gov/snap-all.html>
- [Nmap 2012]** Nmap homepage, <http://www.insecure.com/nmap>
- [Nonnenmacher 1998]** J. Nonnenmacher, E. Biersak, D. Towsley, “Parity-Based Loss Recovery for Reliable Multicast Transmission,” *IEEE/ACM Transactions on Networking*, Vol. 6, No. 4 (Aug. 1998), pp. 349–361.
- [NTIA 1998]** National Telecommunications and Information Administration (NTIA), US Department of Commerce, “Management of Internet names and addresses,” Docket Number: 980212036-8146-02. http://www.ntia.doc.gov/ntiahome/domainname/6_5_98dns.htm
- [O’Dell 2009]** M. O’Dell, “Network Front-End Processors, Yet Again,” *Communications of the ACM*, Vol. 52, No. 6 (June 2009), pp. 46–50.
- [OID Repository 2012]** OID Repository, <http://www.oid-info.com/>
- [ISO 2012]** International Organization for Standardization homepage, <http://www.iso.org/iso/en/ISOOnline.frontpage>
- [OSS 2012]** OSS Nokalva, “ASN.1 Resources,” <http://www.oss.com/asn1/>
- [Padhye 2000]** J. Padhye, V. Firoiu, D. Towsley, J. Kurose, “Modeling TCP Reno Performance: A Simple Model and its Empirical Validation,” *IEEE/ACM Transactions on Networking*, Vol. 8 No. 2 (Apr. 2000), pp. 133–145.
- [Padhye 2001]** J. Padhye, S. Floyd, “On Inferring TCP Behavior,” *Proc. 2001 ACM SIGCOMM* (San Diego, CA, Aug. 2001).
- [Pan 1997]** P. Pan, H. Schulzrinne, “Staged Refresh Timers for RSVP,” *Proc. 2nd Global Internet Conference* (Phoenix, AZ, Dec. 1997).
- [Parekh 1993]** A. Parekh, R. Gallagher, “A generalized processor sharing approach to flow control in integrated services networks: the single-node case,” *IEEE/ACM Transactions on Networking*, Vol. 1, No. 3 (June 1993), pp. 344–357.
- [Partridge 1992]** C. Partridge, S. Pink, “An Implementation of the Revised Internet Stream Protocol (ST-2),” *Journal of Internetworking: Research and Experience*, Vol. 3, No. 1 (Mar. 1992).
- [Partridge 1998]** C. Partridge, et al. “A Fifty Gigabit per second IP Router,” *IEEE/ACM Transactions on Networking*, Vol. 6, No. 3 (Jun. 1998), pp. 237–248.
- [Pathak 2010]** A. Pathak, Y. A. Wang, C. Huang, A. Greenberg, Y. C. Hu, J. Li, K. W. Ross, “Measuring and Evaluating TCP Splitting for Cloud Services,” *Passive and Active Measurement (PAM) Conference* (Zurich, 2010).
- [Paxson 1997]** V. Paxson, “End-to-End Internet Packet Dynamics,” *Proc. 1997 ACM SIGCOMM* (Cannes, France, Sept. 1997).

- [**Perkins 1994**] A. Perkins, “Networking with Bob Metcalfe,” *The Red Herring Magazine* (Nov. 1994).
- [**Perkins 1998**] C. Perkins, O. Hodson, V. Hardman, “A Survey of Packet Loss Recovery Techniques for Streaming Audio,” *IEEE Network Magazine* (Sept./Oct. 1998), pp. 40–47.
- [**Perkins 1998b**] C. Perkins, *Mobile IP: Design Principles and Practice*, Addison-Wesley, Reading, MA, 1998.
- [**Perkins 2000**] C. Perkins, *Ad Hoc Networking*, Addison-Wesley, Reading, MA, 2000.
- [**Perlman 1999**] R. Perlman, *Interconnections: Bridges, Routers, Switches, and Internetworking Protocols*, 2nd ed., Addison-Wesley Professional Computing Series, Reading, MA, 1999.
- [**PGPI 2012**] The International PGP Home Page, <http://www.pgpi.org>
- [**Phifer 2000**] L. Phifer, “The Trouble with NAT,” *The Internet Protocol Journal*, Vol. 3, No. 4 (Dec. 2000), http://www.cisco.com/warp/public/759/ipj_3-4/ipj_3-4_nat.html
- [**Piatek 2007**] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, A. Venkataramani, “Do Incentives Build Robustness in BitTorrent?,” *Proc. NSDI* (2007).
- [**Piatek 2008**] M. Piatek, T. Isdal, A. Krishnamurthy, T. Anderson, “One hop Reputations for Peer-to-peer File Sharing Workloads,” *Proc. NSDI* (2008).
- [**Pickholtz 1982**] R. Pickholtz, D. Schilling, L. Milstein, “Theory of Spread Spectrum Communication—a Tutorial,” *IEEE Transactions on Communications*, Vol. 30, No. 5 (May 1982), pp. 855–884.
- [**PingPlotter 2012**] PingPlotter homepage, <http://www.pingplotter.com>
- [**Piscatello 1993**] D. Piscatello, A. Lyman Chapin, *Open Systems Networking*, Addison-Wesley, Reading, MA, 1993.
- [**Point Topic 2006**] Point Topic Ltd., *World Broadband Statistics Q1 2006*, <http://www.pointtopic.com>
- [**Potaroo 2012**] “Growth of the BGP Table—1994 to Present,” <http://bgp.potaroo.net/>
- [**PPLive 2012**] PPLive homepage, <http://www.pplive.com>
- [**Quagga 2012**] Quagga, “Quagga Routing Suite,” <http://www.quagga.net/>
- [**Quittner 1998**] J. Quittner, M. Slatalla, *Speeding the Net: The Inside Story of Netscape and How it Challenged Microsoft*, Atlantic Monthly Press, 1998.
- [**Quova 2012**] www.quova.com
- [**Raicu 2011**] C. Raicu , S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, M. Handley, “Improving Datacenter Performance and Robustness with Multipath TCP,” *Proc. 2011 ACM SIGCOMM*.
- [**Ramakrishnan 1990**] K. K. Ramakrishnan, R. Jain, “A Binary Feedback Scheme for Congestion Avoidance in Computer Networks,” *ACM Transactions on Computer Systems*, Vol. 8, No. 2 (May 1990), pp. 158–181.
- [**Raman 1999**] S. Raman, S. McCanne, “A Model, Analysis, and Protocol Framework for Soft State-based Communication,” *Proc. 1999 ACM SIGCOMM* (Boston, MA, Aug. 1999).
- [**Raman 2007**] B. Raman, K. Chebrolu, “Experiences in using WiFi for Rural Internet in India,” *IEEE Communications Magazine*, Special Issue on New Directions in Networking Technologies in Emerging Economies (Jan. 2007).

- [**Ramaswami 2010**] R. Ramaswami, K. Sivarajan, G. Sasaki, *Optical Networks: A Practical Perspective*, Morgan Kaufman Publishers, 2010.
- [**Ramjee 1994**] R. Ramjee, J. Kurose, D. Towsley, H. Schulzrinne, “Adaptive Playout Mechanisms for Packetized Audio Applications in Wide-Area Networks,” *Proc. 1994 IEEE INFOCOM*.
- [**Rao 1996**] K. R. Rao and J. J. Hwang, *Techniques and Standards for Image, Video and Audio Coding*, Prentice Hall, Englewood Cliffs, NJ, 1996.
- [**Rao 2011**] A. S. Rao, Y. S. Lim, C. Barakat, A. Legout, D. Towsley, W. Dabbous, “Network Characteristics of Video Streaming Traffic,” *Proc. 2011 ACM CoNEXT* (Tokyo).
- [**RAT 2012**] Robust Audio Tool, <http://www-mice.cs.ucl.ac.uk/multimedia/software/rat/>
- [**Ratnasamy 2001**] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, “A Scalable Content-Addressable Network,” *Proc. 2001 ACM SIGCOMM* (San Diego, CA, Aug. 2001).
- [**Ren 2006**] S. Ren, L. Guo, and X. Zhang, “ASAP: an AS-aware peer-relay protocol for high quality VoIP,” *Proc. 2006 IEEE ICDCS* (Lisboa, Portugal, July 2006).
- [**Rescorla 2001**] E. Rescorla, *SSL and TLS: Designing and Building Secure Systems*, Addison-Wesley, Boston, 2001.
- [**RFC 001**] S. Crocker, “Host Software,” RFC 001 (the *very first* RFC!).
- [**RFC 768**] J. Postel, “User Datagram Protocol,” RFC 768, Aug. 1980.
- [**RFC 789**] E. Rosen, “Vulnerabilities of Network Control Protocols,” RFC 789.
- [**RFC 791**] J. Postel, “Internet Protocol: DARPA Internet Program Protocol Specification,” RFC 791, Sept. 1981.
- [**RFC 792**] J. Postel, “Internet Control Message Protocol,” RFC 792, Sept. 1981.
- [**RFC 793**] J. Postel, “Transmission Control Protocol,” RFC 793, Sept. 1981.
- [**RFC 801**] J. Postel, “NCP/TCP Transition Plan,” RFC 801, Nov. 1981.
- [**RFC 826**] D. C. Plummer, “An Ethernet Address Resolution Protocol—or—Converting Network Protocol Addresses to 48 bit Ethernet Address for Transmission on Ethernet Hardware,” RFC 826, Nov. 1982.
- [**RFC 829**] V. Cerf, “Packet Satellite Technology Reference Sources,” RFC 829, Nov. 1982.
- [**RFC 854**] J. Postel, J. Reynolds, “TELNET Protocol Specification,” RFC 854, May 1993.
- [**RFC 950**] J. Mogul, J. Postel, “Internet Standard Subnetting Procedure,” RFC 950, Aug. 1985.
- [**RFC 959**] J. Postel and J. Reynolds, “File Transfer Protocol (FTP),” RFC 959, Oct. 1985.
- [**RFC 977**] B. Kantor, P. Lapsley, “Network News Transfer Protocol,” RFC 977, Feb. 1986.
- [**RFC 1028**] J. Davin, J.D. Case, M. Fedor, M. Schoffstall, “A Simple Gateway Monitoring Protocol,” RFC 1028, Nov. 1987.
- [**RFC 1034**] P. V. Mockapetris, “Domain Names—Concepts and Facilities,” RFC 1034, Nov. 1987.
- [**RFC 1035**] P. Mockapetris, “Domain Names—Implementation and Specification,” RFC 1035, Nov. 1987.
- [**RFC 1058**] C. L. Hendrick, “Routing Information Protocol,” RFC 1058, June 1988.

- [RFC 1071] R. Braden, D. Borman, and C. Partridge, “Computing The Internet Checksum,” RFC 1071, Sept. 1988.
- [RFC 1075] D. Waitzman, C. Partridge, S. Deering, “Distance Vector Multicast Routing Protocol,” RFC 1075, Nov. 1988.
- [RFC 1112] S. Deering, “Host Extension for IP Multicasting,” RFC 1112, Aug. 1989.
- [RFC 1122] R. Braden, “Requirements for Internet Hosts—Communication Layers,” RFC 1122, Oct. 1989.
- [RFC 1123] R. Braden, ed., “Requirements for Internet Hosts—Application and Support,” RFC-1123, Oct. 1989.
- [RFC 1142] D. Oran, “OSI IS-IS Intra-Domain Routing Protocol,” RFC 1142, Feb. 1990.
- [RFC 1190] C. Topolcic, “Experimental Internet Stream Protocol: Version 2 (ST-II),” RFC 1190, Oct. 1990.
- [RFC 1191] J. Mogul, S. Deering, “Path MTU Discovery,” RFC 1191, Nov. 1990.
- [RFC 1213] K. McCloghrie, M. T. Rose, “Management Information Base for Network Management of TCP/IP-based internets: MIB-II,” RFC 1213, Mar. 1991.
- [RFC 1256] S. Deering, “ICMP Router Discovery Messages,” RFC 1256, Sept. 1991.
- [RFC 1320] R. Rivest, “The MD4 Message-Digest Algorithm,” RFC 1320, Apr. 1992.
- [RFC 1321] R. Rivest, “The MD5 Message-Digest Algorithm,” RFC 1321, Apr. 1992.
- [RFC 1323] V. Jacobson, S. Braden, D. Borman, “TCP Extensions for High Performance,” RFC 1323, May 1992.
- [RFC 1422] S. Kent, “Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management,” RFC 1422.
- [RFC 1546] C. Partridge, T. Mendez, W. Milliken, “Host Anycasting Service,” RFC 1546, 1993.
- [RFC 1547] D. Perkins, “Requirements for an Internet Standard Point-to-Point Protocol,” RFC 1547, Dec. 1993.
- [RFC 1584] J. Moy, “Multicast Extensions to OSPF,” RFC 1584, Mar. 1994.
- [RFC 1633] R. Braden, D. Clark, S. Shenker, “Integrated Services in the Internet Architecture: an Overview,” RFC 1633, June 1994.
- [RFC 1636] R. Braden, D. Clark, S. Crocker, C. Huitema, “Report of IAB Workshop on Security in the Internet Architecture,” RFC 1636, Nov. 1994.
- [RFC 1661] W. Simpson (ed.), “The Point-to-Point Protocol (PPP),” RFC 1661, July 1994.
- [RFC 1662] W. Simpson (ed.), “PPP in HDLC-Like Framing,” RFC 1662, July 1994.
- [RFC 1700] J. Reynolds and J. Postel, “Assigned Numbers,” RFC 1700, Oct. 1994.
- [RFC 1752] S. Bradner, A. Mankin, “The Recommendations for the IP Next Generation Protocol,” RFC 1752, Jan. 1995.
- [RFC 1918] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, E. Lear, “Address Allocation for Private Internets,” RFC 1918, Feb. 1996.
- [RFC 1930] J. Hawkinson, T. Bates, “Guidelines for Creation, Selection, and Registration of an Autonomous System (AS),” RFC 1930, Mar. 1996.
- [RFC 1938] N. Haller, C. Metz, “A One-Time Password System,” RFC 1938, May 1996.
- [RFC 1939] J. Myers and M. Rose, “Post Office Protocol—Version 3,” RFC 1939, May 1996.

- [RFC 1945] T. Berners-Lee, R. Fielding, H. Frystyk, “Hypertext Transfer Protocol—HTTP/1.0,” RFC 1945, May 1996.
- [RFC 2003] C. Perkins, “IP Encapsulation within IP,” RFC 2003, Oct. 1996.
- [RFC 2004] C. Perkins, “Minimal Encapsulation within IP,” RFC 2004, Oct. 1996.
- [RFC 2018] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, “TCP Selective Acknowledgment Options,” RFC 2018, Oct. 1996.
- [RFC 2050] K. Hubbard, M. Kosters, D. Conrad, D. Karrenberg, J. Postel, “Internet Registry IP Allocation Guidelines,” RFC 2050, Nov. 1996.
- [RFC 2104] H. Krawczyk, M. Bellare, R. Canetti, “HMAC: Keyed-Hashing for Message Authentication,” RFC 2104, Feb. 1997.
- [RFC 2131] R. Droms, “Dynamic Host Configuration Protocol,” RFC 2131, Mar. 1997.
- [RFC 2136] P. Vixie, S. Thomson, Y. Rekhter, J. Bound, “Dynamic Updates in the Domain Name System,” RFC 2136, Apr. 1997.
- [RFC 2153] W. Simpson, “PPP Vendor Extensions,” RFC 2153, May 1997.
- [RFC 2205] R. Braden, Ed., L. Zhang, S. Berson, S. Herzog, S. Jamin, “Resource ReSerVation Protocol (RSVP)—Version 1 Functional Specification,” RFC 2205, Sept. 1997.
- [RFC 2210] J. Wroclawski, “The Use of RSVP with IETF Integrated Services,” RFC 2210, Sept. 1997.
- [RFC 2211] J. Wroclawski, “Specification of the Controlled-Load Network Element Service,” RFC 2211, Sept. 1997.
- [RFC 2215] S. Shenker, J. Wroclawski, “General Characterization Parameters for Integrated Service Network Elements,” RFC 2215, Sept. 1997.
- [RFC 2326] H. Schulzrinne, A. Rao, R. Lanphier, “Real Time Streaming Protocol (RTSP),” RFC 2326, Apr. 1998.
- [RFC 2328] J. Moy, “OSPF Version 2,” RFC 2328, Apr. 1998.
- [RFC 2420] H. Kummert, “The PPP Triple-DES Encryption Protocol (3DESE),” RFC 2420, Sept. 1998.
- [RFC 2453] G. Malkin, “RIP Version 2,” RFC 2453, Nov. 1998.
- [RFC 2460] S. Deering, R. Hinden, “Internet Protocol, Version 6 (IPv6) Specification,” RFC 2460, Dec. 1998.
- [RFC 2475] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss, “An Architecture for Differentiated Services,” RFC 2475, Dec. 1998.
- [RFC 2578] K. McCloghrie, D. Perkins, J. Schoenwaelder, “Structure of Management Information Version 2 (SMIV2),” RFC 2578, Apr. 1999.
- [RFC 2579] K. McCloghrie, D. Perkins, J. Schoenwaelder, “Textual Conventions for SMIV2,” RFC 2579, Apr. 1999.
- [RFC 2580] K. McCloghrie, D. Perkins, J. Schoenwaelder, “Conformance Statements for SMIV2,” RFC 2580, Apr. 1999.
- [RFC 2597] J. Heinanen, F. Baker, W. Weiss, J. Wroclawski, “Assured Forwarding PHB Group,” RFC 2597, June 1999.
- [RFC 2616] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, R. Fielding, “Hypertext Transfer Protocol—HTTP/1.1,” RFC 2616, June 1999.

- [RFC 2663] P. Srisuresh, M. Holdrege, “IP Network Address Translator (NAT) Terminology and Considerations,” RFC 2663.
- [RFC 2702] D. Awdutche, J. Malcolm, J. Agogbuia, M. O’Dell, J. McManus, “Requirements for Traffic Engineering Over MPLS,” RFC 2702, Sept. 1999.
- [RFC 2827] P. Ferguson, D. Senie, “Network Ingress Filtering: Defeating Denial of Service Attacks which Employ IP Source Address Spoofing,” RFC 2827, May 2000.
- [RFC 2865] C. Rigney, S. Willens, A. Rubens, W. Simpson, “Remote Authentication Dial In User Service (RADIUS),” RFC 2865, June 2000.
- [RFC 2961] L. Berger, D. Gan, G. Swallow, P. Pan, F. Tommasi, S. Molendini, “RSVP Refresh Overhead Reduction Extensions,” RFC 2961, Apr. 2001.
- [RFC 3007] B. Wellington, “Secure Domain Name System (DNS) Dynamic Update,” RFC 3007, Nov. 2000.
- [RFC 3022] P. Srisuresh, K. Egevang, “Traditional IP Network Address Translator (Traditional NAT),” RFC 3022, Jan. 2001.
- [RFC 3022] P. Srisuresh, K. Egevang, “Traditional IP Network Address Translator (Traditional NAT),” RFC 3022, Jan. 2001.
- [RFC 3031] E. Rosen, A. Viswanathan, R. Callon, “Multiprotocol Label Switching Architecture,” RFC 3031, Jan. 2001.
- [RFC 3032] E. Rosen, D. Tappan, G. Fedorkow, Y. Rekhter, D. Farinacci, T. Li, A. Conta, “MPLS Label Stack Encoding,” RFC 3032, Jan. 2001.
- [RFC 3052] M. Eder, S. Nag, “Service Management Architectures Issues and Review,” RFC 3052, Jan. 2001.
- [RFC 3139] L. Sanchez, K. McCloghrie, J. Saperia, “Requirements for Configuration Management of IP-Based Networks,” RFC 3139, June 2001.
- [RFC 3168] K. Ramakrishnan, S. Floyd, D. Black, “The Addition of Explicit Congestion Notification (ECN) to IP,” RFC 3168, Sept. 2001.
- [RFC 3209] D. Awdutche, L. Berger, D. Gan, T. Li, V. Srinivasan, G. Swallow, “RSVP-TE: Extensions to RSVP for LSP Tunnels,” RFC 3209, Dec. 2001.
- [RFC 3221] G. Huston, “Commentary on Inter-Domain Routing in the Internet,” RFC 3221, Dec. 2001.
- [RFC 3232] J. Reynolds, “Assigned Numbers: RFC 1700 is Replaced by an On-line Database,” RFC 3232, Jan. 2002.
- [RFC 3246] B. Davie, A. Charny, J.C.R. Bennet, K. Benson, J.Y. Le Boudec, W. Courtney, S. Davari, V. Firoiu, D. Stiliadis, “An Expedited Forwarding PHB (Per-Hop Behavior),” RFC 3246, Mar. 2002.
- [RFC 3260] D. Grossman, “New Terminology and Clarifications for DiffServ,” RFC 3260, Apr. 2002.
- [RFC 3261] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler, “SIP: Session Initiation Protocol,” RFC 3261, July 2002.
- [RFC 3272] J. Boyle, V. Gill, A. Hannan, D. Cooper, D. Awdutche, B. Christian, W.S. Lai, “Overview and Principles of Internet Traffic Engineering,” RFC 3272, May 2002.
- [RFC 3286] L. Ong, J. Yoakum, “An Introduction to the Stream Control Transmission Protocol (SCTP),” RFC 3286, May 2002.

- [RFC 3346] J. Boyle, V. Gill, A. Hannan, D. Cooper, D. Awduche, B. Christian, W. S. Lai, “Applicability Statement for Traffic Engineering with MPLS,” RFC 3346, Aug. 2002.
- [RFC 3376] B. Cain, S. Deering, I. Kouvelas, B. Fenner, A. Thyagarajan, “Internet Group Management Protocol, Version 3,” RFC 3376, Oct. 2002.
- [RFC 3390] M. Allman, S. Floyd, C. Partridge, “Increasing TCP’s Initial Window,” RFC 3390, Oct. 2002.
- [RFC 3410] J. Case, R. Mundy, D. Partain, “Introduction and Applicability Statements for Internet Standard Management Framework,” RFC 3410, Dec. 2002.
- [RFC 3411] D. Harrington, R. Presuhn, B. Wijnen, “An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks,” RFC 3411, Dec. 2002.
- [RFC 3414] U. Blumenthal and B. Wijnen, “User-based Security Model (USM) for Version 3 of the Simple Network Management Protocol (SNMPv3),” RFC 3414, December 2002.
- [RFC 3415] B. Wijnen, R. Presuhn, K. McCloghrie, “View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP),” RFC 3415, Dec. 2002.
- [RFC 3416] R. Presuhn, J. Case, K. McCloghrie, M. Rose, S. Waldbusser, “Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP),” Dec. 2002.
- [RFC 3439] R. Bush and D. Meyer, “Some internet architectural guidelines and philosophy,” RFC 3439, Dec. 2003.
- [RFC 3447] J. Jonsson, B. Kaliski, “Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1,” RFC 3447, Feb. 2003.
- [RFC 3468] L. Andersson, G. Swallow, “The Multiprotocol Label Switching (MPLS) Working Group Decision on MPLS Signaling Protocols,” RFC 3468, Feb. 2003.
- [RFC 3469] V. Sharma, Ed., F. Hellstrand, Ed, “Framework for Multi-Protocol Label Switching (MPLS)-based Recovery,” RFC 3469, Feb. 2003. <ftp://ftp.rfc-editor.org/in-notes/rfc3469.txt>
- [RFC 3501] M. Crispin, “Internet Message Access Protocol—Version 4rev1,” RFC 3501, Mar. 2003.
- [RFC 3550] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, “RTP: A Transport Protocol for Real-Time Applications,” RFC 3550, July 2003.
- [RFC 3569] S. Bhattacharyya (ed.), “An Overview of Source-Specific Multicast (SSM),” RFC 3569, July 2003.
- [RFC 3588] P. Calhoun, J. Loughney, E. Guttman, G. Zorn, J. Arkko, “Diameter Base Protocol,” RFC 3588, Sept. 2003.
- [RFC 3618] B. Fenner, D. Meyer, Ed., “Multicast Source Discovery Protocol (MSDP),” RFC 3618, Oct. 2003.
- [RFC 3649] S. Floyd, “High Speed TCP for Large Congestion Windows,” RFC 3649, Dec. 2003.
- [RFC 3748] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, H. Levkowetz, Ed., “Extensible Authentication Protocol (EAP),” RFC 3748, June 2004.
- [RFC 3782] S. Floyd, T. Henderson, A. Gurkov, “The NewReno Modification to TCP’s Fast Recovery Algorithm,” RFC 3782, Apr. 2004.

- [RFC 3973] A. Adams, J. Nicholas, W. Siadak, “Protocol Independent Multicast—Dense Mode (PIM-DM): Protocol Specification (Revised),” RFC 3973, Jan. 2005.
- [RFC 4022] R. Raghunarayanan, Ed., “Management Information Base for the Transmission Control Protocol (TCP),” RFC 4022, Mar. 2005.
- [RFC 4113] B. Fenner, J. Flick, “Management Information Base for the User Datagram Protocol (UDP),” RFC 4113, June 2005.
- [RFC 4213] E. Nordmark, R. Gilligan, “Basic Transition Mechanisms for IPv6 Hosts and Routers,” RFC 4213, Oct. 2005.
- [RFC 4271] Y. Rekhter, T. Li, S. Hares, Ed., “A Border Gateway Protocol 4 (BGP-4),” RFC 4271, Jan. 2006.
- [RFC 4272] S. Murphy, “BGP Security Vulnerabilities Analysis,” RFC 4274, Jan. 2006.
- [RFC 4274] Meyer, D. and K. Patel, “BGP-4 Protocol Analysis”, RFC 4274, January 2006.
- [RFC 4291] R. Hinden, S. Deering, “IP Version 6 Addressing Architecture,” RFC 4291, February 2006.
- [RFC 4293] S. Routhier, Ed. “Management Information Base for the Internet Protocol (IP),” RFC 4293, Apr. 2006.
- [RFC 4301] S. Kent, K. Seo, “Security Architecture for the Internet Protocol,” RFC 4301, Dec. 2005.
- [RFC 4302] S. Kent, “IP Authentication Header,” RFC 4302, Dec. 2005.
- [RFC 4303] S. Kent, “IP Encapsulating Security Payload (ESP),” RFC 4303, Dec. 2005.
- [RFC 4305] D. Eastlake, “Cryptographic Algorithm Implementation Requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH),” RFC 4305, Dec. 2005.
- [RFC 4340] E. Kohler, M. Handley, S. Floyd, “Datagram Congestion Control Protocol (DCCP),” RFC 4340, Mar. 2006.
- [RFC 4443] A. Conta, S. Deering, M. Gupta, Ed., “Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification,” RFC 4443, Mar. 2006.
- [RFC 4346] T. Dierks, E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.1,” RFC 4346, Apr. 2006.
- [RFC 4502] S. Waldbusser, “Remote Network Monitoring Management Information Base Version 2,” RFC 4502, May 2006.
- [RFC 4514] K. Zeilenga, Ed., “Lightweight Directory Access Protocol (LDAP): String Representation of Distinguished Names,” RFC 4514, June 2006.
- [RFC 4601] B. Fenner, M. Handley, H. Holbrook, I. Kouvelas, “Protocol Independent Multicast—Sparse Mode (PIM-SM): Protocol Specification (Revised),” RFC 4601, Aug. 2006.
- [RFC 4607] H. Holbrook, B. Cain, “Source-Specific Multicast for IP,” RFC 4607, Aug. 2006.
- [RFC 4611] M. McBride, J. Meylor, D. Meyer, “Multicast Source Discovery Protocol (MSDP) Deployment Scenarios,” RFC 4611, Aug. 2006.
- [RFC 4632] V. Fuller, T. Li, “Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan,” RFC 4632, Aug. 2006.

- [RFC 4960] R. Stewart, ed., “Stream Control Transmission Protocol,” RFC 4960, Sept. 2007.
- [RFC 4987] W. Eddy, “TCP SYN Flooding Attacks and Common Mitigations,” RFC 4987, Aug. 2007.
- [RFC 5000] RFC editor, “Internet Official Protocol Standards,” RFC 5000, May 2008.
- [RFC 5109] A. Li (ed.), “RTP Payload Format for Generic Forward Error Correction,” RFC 5109, Dec. 2007.
- [RFC 5110] P. Savola, “Overview of the Internet Multicast Routing Architecture,” RFC 5110, Jan. 2008.
- [RFC 5216] D. Simon, B. Aboba, R. Hurst, “The EAP-TLS Authentication Protocol,” RFC 5216, Mar. 2008.
- [RFC 5218] D. Thaler, B. Aboba, “What Makes for a Successful Protocol?,” RFC 5218, July 2008.
- [RFC 5321] J. Klensin, “Simple Mail Transfer Protocol,” RFC 5321, Oct. 2008.
- [RFC 5322] P. Resnick, Ed., “Internet Message Format,” RFC 5322, Oct. 2008.
- [RFC 5348] S. Floyd, M. Handley, J. Padhye, J. Widmer, “TCP Friendly Rate Control (TFRC): Protocol Specification,” RFC 5348, Sept. 2008.
- [RFC 5411] J Rosenberg, “A Hitchhiker’s Guide to the Session Initiation Protocol (SIP),” RFC 5411, Feb. 2009.
- [RFC 5681] M. Allman, V. Paxson, E. Blanton, “TCP Congestion Control,” RFC 5681, Sept. 2009.
- [RFC 5944] C. Perkins, Ed., “IP Mobility Support for IPv4, Revised,” RFC 5944, November 2010.
- [RFC 5996] C. Kaufman, P. Hoffman, Y. Nir, P. Eronen, “Internet Key Exchange Protocol Version 2 (IKEv2),” RFC 5996, Sept. 2010.
- [RFC 6071] S. Frankel, S. Krishnan, “IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap,” RFC 6071, Feb. 2011.
- [RFC 6265] A Barth, “HTTP State Management Mechanism,” RFC 6265, Apr. 2011.
- [RFC 6298] V. Paxson, M. Allman, J. Chu, M. Sargent, “Computing TCP’s Retransmission Timer,” RFC 6298, June 2011.
- [Rhee 1998] I. Rhee, “Error Control Techniques for Interactive Low-Bit Rate Video Transmission over the Internet,” *Proc. 1998 ACM SIGCOMM* (Vancouver BC, Aug. 1998).
- [Roberts 1967] L. Roberts, T. Merrill, “Toward a Cooperative Network of Time-Shared Computers,” *AFIPS Fall Conference* (Oct. 1966).
- [Roberts 2004] J. Roberts, “Internet Traffic, QoS and Pricing,” *Proc. 2004 IEEE INFOCOM*, Vol. 92, No. 9 (Sept. 2004), pp. 1389–1399.
- [Rodriguez 2010] R. Rodrigues, P. Druschel, “Peer-to-Peer Systems,” *Communications of the ACM*, Vol. 53, No. 10 (Oct. 2010), pp. 72–82.
- [Rohde 2008] Rohde and Schwarz, “UMTS Long Term Evolution (LTE) Technology Introduction,” Application Note 1MA111.
- [Rom 1990] R. Rom, M. Sidi, *Multiple Access Protocols: Performance and Analysis*, Springer-Verlag, New York, 1990.
- [Root Servers 2012] Root Servers homepage, <http://www.root-servers.org/>

- [**Rose 1996**] M. Rose, *The Simple Book: An Introduction to Internet Management, Revised Second Edition*, Prentice Hall, Englewood Cliffs, NJ, 1996.
- [**Ross 1995**] K. W. Ross, *Multiservice Loss Models for Broadband Telecommunication Networks*, Springer, Berlin, 1995.
- [**Rowston 2001**] A. Rowston, P. Druschel, “Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems,” *Proc. 2001 IFIP/ACM Middleware* (Heidelberg, Germany, 2001).
- [**RSA 1978**] R. Rivest, A. Shamir, L. Adelman, “A Method for Obtaining Digital Signatures and Public-key Cryptosystems,” *Communications of the ACM*, Vol. 21, No. 2 (Feb. 1978), pp. 120–126.
- [**RSA Fast 2012**] RSA Laboratories, “How Fast is RSA?” <http://www.rsa.com/rsalabs/node.asp?id=2215>
- [**RSA Key 2012**] RSA Laboratories, “How large a key should be used in the RSA Crypto system?” <http://www.rsa.com/rsalabs/node.asp?id=2218>
- [**Rubenstein 1998**] D. Rubenstein, J. Kurose, D. Towsley, “Real-Time Reliable Multicast Using Proactive Forward Error Correction,” *Proceedings of NOSSDAV ’98* (Cambridge, UK, July 1998).
- [**Rubin 2001**] A. Rubin, *White-Hat Security Arsenal: Tackling the Threats*, Addison-Wesley, 2001.
- [**Ruiz-Sánchez 2001**] M. Ruiz-Sánchez, E. Biersack, W. Dabbous, “Survey and Taxonomy of IP Address Lookup Algorithms,” *IEEE Network Magazine*, Vol. 15, No. 2 (Mar./Apr. 2001), pp. 8–23.
- [**Saltzer 1984**] J. Saltzer, D. Reed, D. Clark, “End-to-End Arguments in System Design,” *ACM Transactions on Computer Systems (TOCS)*, Vol. 2, No. 4 (Nov. 1984).
- [**Sandvine 2011**] “Global Internet Phenomena Report, Spring 2011,” <http://www.sandvine.com/news/global broadband trends.asp>, 2011.
- [**Sardar 2006**] B. Sardar, D. Saha, “A Survey of TCP Enhancements for Last-Hop Wireless Networks,” *IEEE Commun. Surveys and Tutorials*, Vol. 8, No. 3 (2006), pp. 20–34.
- [**Saroiu 2002**] S. Saroiu, P.K. Gummadi, S.D. Gribble, “A Measurement Study of Peer-to-Peer File Sharing Systems,” *Proc. of Multimedia Computing and Networking (MMCN)* (2002).
- [**Saroiu 2002b**] S. Saroiu, K. P. Gummadi, R. J. Dunn, S. D. Gribble, and H. M. Levy, “An Analysis of Internet Content Delivery Systems,” *USENIX OSDI* (2002).
- [**Saydam 1996**] T. Saydam, T. Magedanz, “From Networks and Network Management into Service and Service Management,” *Journal of Networks and System Management*, Vol. 4, No. 4 (Dec. 1996), pp. 345–348.
- [**Schiller 2003**] J. Schiller, *Mobile Communications* 2nd edition, Addison Wesley, 2003.
- [**Schneier 1995**] B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, John Wiley and Sons, 1995.
- [**Schulzrinne 1997**] H. Schulzrinne, “A Comprehensive Multimedia Control Architecture for the Internet,” *NOSSDAV’97 (Network and Operating System Support for Digital Audio and Video)* (St. Louis, MO, May 1997).
- [**Schulzrinne-RTP 2012**] Henning Schulzrinne’s RTP site, <http://www.cs.columbia.edu/~hgs/rtp>

- [Schulzrinne-RTSP 2012]** Henning Schulzrinne's RTSP site, <http://www.cs.columbia.edu/~hgs/rtsp>
- [Schulzrinne-SIP 2012]** Henning Schulzrinne's SIP site, <http://www.cs.columbia.edu/~hgs/sip>
- [Schwartz 1977]** M. Schwartz, *Computer-Communication Network Design and Analysis*, Prentice-Hall, Englewood Cliffs, N.J., 1997.
- [Schwartz 1980]** M. Schwartz, *Information, Transmission, Modulation, and Noise*, McGraw Hill, New York, NY 1980.
- [Schwartz 1982]** M. Schwartz, "Performance Analysis of the SNA Virtual Route Pacing Control," *IEEE Transactions on Communications*, Vol. 30, No. 1 (Jan. 1982), pp. 172–184.
- [Scourias 2012]** J. Scourias, "Overview of the Global System for Mobile Communications: GSM." <http://www.privateline.com/PCS/GSM0.html>
- [Segaller 1998]** S. Segaller, *Nerds 2.0.1, A Brief History of the Internet*, TV Books, New York, 1998.
- [Shacham 1990]** N. Shacham, P. McKenney, "Packet Recovery in High-Speed Networks Using Coding and Buffer Management," *Proc. 1990 IEEE INFOCOM* (San Francisco, CA, Apr. 1990), pp. 124–131.
- [Shaikh 2001]** A. Shaikh, R. Tewari, M. Agrawal, "On the Effectiveness of DNS-based Server Selection," *Proc. 2001 IEEE INFOCOM*.
- [Sharma 2003]** P. Sharma, E. Perry, R. Malpani, "IP Multicast Operational Network management: Design, Challenges, and Experiences," *IEEE Network Magazine* (Mar. 2003), pp. 49–55.
- [Singh 1999]** S. Singh, *The Code Book: The Evolution of Secrecy from Mary, Queen of Scots to Quantum Cryptography*, Doubleday Press, 1999.
- [SIP Software 2012]** H. Schulzrinne Software Package site, <http://www.cs.columbia.edu/IRT/software>
- [Skoudis 2004]** E. Skoudis, L. Zeltser, *Malware: Fighting Malicious Code*, Prentice Hall, 2004.
- [Skoudis 2006]** E. Skoudis, T. Liston, *Counter Hack Reloaded: A Step-by-Step Guide to Computer Attacks and Effective Defenses* (2nd Edition), Prentice Hall, 2006.
- [Skype 2012]** Skype homepage, www.skype.com
- [SMIL 2012]** W3C Synchronized Multimedia homepage, <http://www.w3.org/AudioVideo>
- [Smith 2009]** J. Smith, "Fighting Physics: A Tough Battle," *Communications of the ACM*, Vol. 52, No. 7 (July 2009), pp. 60–65.
- [Snort 2012]** Sourcefire Inc., Snort homepage, <http://www.snort.org/>
- [Solari 1997]** S. J. Solari, *Digital Video and Audio Compression*, McGraw Hill, New York, NY, 1997.
- [Solensky 1996]** F. Solensky, "IPv4 Address Lifetime Expectations," in *IPng: Internet Protocol Next Generation* (S. Bradner, A. Mankin, ed.), Addison-Wesley, Reading, MA, 1996.
- [Spragins 1991]** J. D. Spragins, *Telecommunications Protocols and Design*, Addison-Wesley, Reading, MA, 1991.
- [Srikant 2004]** R. Srikant, *The Mathematics of Internet Congestion Control*, Birkhauser, 2004
- [Sripanidkulchai 2004]** K. Sripanidkulchai, B. Maggs, and H. Zhang, "An analysis of live streaming workloads on the Internet," *Proc. 2004 ACM Internet Measurement Conference* (Taormina, Sicily, Italy), pp. 41–54.

- [**Stallings 1993**] W. Stallings, *SNMP, SNMP v2, and CMIP The Practical Guide to Network Management Standards*, Addison-Wesley, Reading, MA, 1993.
- [**Stallings 1999**] W. Stallings, *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*, Addison-Wesley, Reading, MA, 1999.
- [**Steinder 2002**] M. Steinder, A. Sethi, “Increasing robustness of fault localization through analysis of lost, spurious, and positive symptoms,” *Proc. 2002 IEEE INFOCOM*.
- [**Stevens 1990**] W. R. Stevens, *Unix Network Programming*, Prentice-Hall, Englewood Cliffs, NJ.
- [**Stevens 1994**] W. R. Stevens, *TCP/IP Illustrated, Vol. 1: The Protocols*, Addison-Wesley, Reading, MA, 1994.
- [**Stevens 1997**] W.R. Stevens, *Unix Network Programming, Volume 1: Networking APIs-Sockets and XTI*, 2nd edition, Prentice-Hall, Englewood Cliffs, NJ, 1997.
- [**Stewart 1999**] J. Stewart, *BGP4: Interdomain Routing in the Internet*, Addison-Wesley, 1999.
- [**Stoica 2001**] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, H. Balakrishnan, “Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications,” *Proc. 2001 ACM SIGCOMM* (San Diego, CA, Aug. 2001).
- [**Stone 1998**] J. Stone, M. Greenwald, C. Partridge, J. Hughes, “Performance of Checksums and CRC’s Over Real Data,” *IEEE/ACM Transactions on Networking*, Vol. 6, No. 5 (Oct. 1998), pp. 529–543.
- [**Stone 2000**] J. Stone, C. Partridge, “When Reality and the Checksum Disagree,” *Proc. 2000 ACM SIGCOMM* (Stockholm, Sweden, Aug. 2000).
- [**Strayer 1992**] W. T. Strayer, B. Dempsey, A. Weaver, *XTP: The Xpress Transfer Protocol*, Addison-Wesley, Reading, MA, 1992.
- [**Stubblefield 2002**] A. Stubblefield, J. Ioannidis, A. Rubin, “Using the Fluhrer, Mantin, and Shamir Attack to Break WEP,” *Proceedings of 2002 Network and Distributed Systems Security Symposium* (2002), pp. 17–22.
- [**Subramanian 2000**] M. Subramanian, *Network Management: Principles and Practice*, Addison-Wesley, Reading, MA, 2000.
- [**Subramanian 2002**] L. Subramanian, S. Agarwal, J. Rexford, R. Katz, “Characterizing the Internet Hierarchy from Multiple Vantage Points,” *Proc. 2002 IEEE INFOCOM*.
- [**Sundaresan 2006**] K. Sundaresan, K. Papagiannaki, “The Need for Cross-layer Information in Access Point Selection,” *Proc. 2006 ACM Internet Measurement Conference* (Rio De Janeiro, Oct. 2006).
- [**Su 2006**] A.-J. Su, D. Choffnes, A. Kuzmanovic, and F. Bustamante, “Drafting Behind Akamai” *Proc. 2006 ACM SIGCOMM*.
- [**Suh 2006**] K. Suh, D. R. Figueiredo, J. Kurose and D. Towsley, “Characterizing and detecting relayed traffic: A case study using Skype,” *Proc. 2006 IEEE INFOCOM* (Barcelona, Spain, Apr. 2006).
- [**Sunshine 1978**] C. Sunshine, Y. Dalal, “Connection Management in Transport Protocols,” *Computer Networks*, North-Holland, Amsterdam, 1978.
- [**Tariq 2008**] M. Tariq, A. Zeitoun, V. Valancius, N. Feamster, M. Ammar, “Answering What-If Deployment and Configuration Questions with WISE,” *Proc. 2008 ACM SIGCOMM* (Aug. 2008).

- [TechnOnLine 2012] TechOnLine, “Protected Wireless Networks,” online webcast tutorial, http://www.techononline.com/community/tech_topic/internet/21752
- [Teixeira 2006] R. Teixeira and J. Rexford, “Managing Routing Disruptions in Internet Service Provider Networks,” *IEEE Communications Magazine* (Mar. 2006).
- [Thaler 1997] D. Thaler and C. Ravishankar, “Distributed Center-Location Algorithms,” *IEEE Journal on Selected Areas in Communications*, Vol. 15, No. 3 (Apr. 1997), pp. 291–303.
- [Think 2012] Technical History of Network Protocols, “Cyclades,” <http://www.cs.utexas.edu/users/chris/think/Cyclades/index.shtml>
- [Tian 2012] Y. Tian, R. Dey, Y. Liu, K. W. Ross, “China’s Internet: Topology Mapping and Geolocating,” *IEEE INFOCOM Mini-Conference 2012* (Orlando, FL, 2012).
- [Tobagi 1990] F. Tobagi, “Fast Packet Switch Architectures for Broadband Integrated Networks,” *Proc. 1990 IEEE INFOCOM*, Vol. 78, No. 1 (Jan. 1990), pp. 133–167.
- [TOR 2012] Tor: Anonymity Online, <http://www.torproject.org>
- [Torres 2011] R. Torres, A. Finamore, J. R. Kim, M. M. Munafó, S. Rao, “Dissecting Video Server Selection Strategies in the YouTube CDN,” *Proc. 2011 Int. Conf. on Distributed Computing Systems*.
- [Turner 1988] J. S. Turner “Design of a Broadcast packet switching network,” *IEEE Transactions on Communications*, Vol. 36, No. 6 (June 1988), pp. 734–743.
- [Turner 2012] B. Turner, “2G, 3G, 4G Wireless Tutorial,” <http://blogs.nmscommunications.com/communications/2008/10/2g-3g-4g-wireless-tutorial.html>
- [UPnP Forum 2012] UPnP Forum homepage, <http://www.upnp.org/>
- [van der Berg 2008] R. van der Berg, “How the ‘Net works: an introduction to peering and transit,” <http://arstechnica.com/guides/other/peering-and-transit.ars>
- [Varghese 1997] G. Varghese, A. Lauck, “Hashed and Hierarchical Timing Wheels: Efficient Data Structures for Implementing a Timer Facility,” *IEEE/ACM Transactions on Networking*, Vol. 5, No. 6 (Dec. 1997), pp. 824–834.
- [Vasudevan 2012] S. Vasudevan, C. Diot, J. Kurose, D. Towsley, “Facilitating Access Point Selection in IEEE 802.11 Wireless Networks,” *Proc. 2005 ACM Internet Measurement Conference*, (San Francisco CA, Oct. 2005).
- [Verizon FIOS 2012] Verizon, “Verizon FiOS Internet: FAQ,” <http://www22.verizon.com/residential/fiosinternet/faq/faq.htm>
- [Verizon SLA 2012] Verizon, “Global Latency and Packet Delivery SLA,” http://www.verizonbusiness.com/terms/global_latency_sla.xml
- [Verma 2001] D. C. Verma, *Content Distribution Networks: An Engineering Approach*, John Wiley, 2001.
- [Villamizar 1994] C. Villamizar, C. Song. “High performance tcp in ansnet,” *ACM SIGCOMM Computer Communications Review*, Vol. 24, No. 5 (1994), pp. 45–60.
- [Viterbi 1995] A. Viterbi, *CDMA: Principles of Spread Spectrum Communication*, Addison-Wesley, Reading, MA, 1995.
- [Vixie 2009] P. Vixie, “What DNS Is Not,” *Communications of the ACM*, Vol. 52, No. 12 (Dec. 2009), pp. 43–47.
- [W3C 1995] The World Wide Web Consortium, “A Little History of the World Wide Web” (1995), <http://www.w3.org/History.html>

- [**Wakeman 1992**] I. Wakeman, J. Crowcroft, Z. Wang, D. Sirovica, “Layering Considered Harmful,” *IEEE Network* (Jan. 1992), pp. 20–24.
- [**Waldrop 2007**] M. Waldrop, “Data Center in a Box,” *Scientific American* (July 2007).
- [**Walker 2000**] J. Walker, “IEEE P802.11 Wireless LANs, Unsafe at Any Key Size; An Analysis of the WEP Encapsulation,” Oct. 2000, <http://www.drizzle.com/~aboba/IEEE/0-362.zip>
- [**Wall 1980**] D. Wall, *Mechanisms for Broadcast and Selective Broadcast*, Ph.D. thesis, Stanford University, June 1980.
- [**Wang 2004**] B. Wang, J. Kurose, P. Shenoy, D. Towsley, “Multimedia Streaming via TCP: An Analytic Performance Study,” *Proc. 2004 ACM Multimedia Conference* (New York, NY, Oct. 2004).
- [**Wang 2008**] B. Wang, J. Kurose, P. Shenoy, D. Towsley, “Multimedia Streaming via TCP: An Analytic Performance Study,” *ACM Transactions on Multimedia Computing Communications and Applications (TOMCCAP)*, Vol. 4, No. 2 (Apr. 2008), pp. 16:1–22.
- [**Wang 2010**] G. Wang, D. G. Andersen, M. Kaminsky, K. Papagiannaki, T. S. E. Ng, M. Kozuch, M. Ryan, “c-Through: Part-time Optics in Data Centers,” *Proc. 2010 ACM SIGCOMM*.
- [**Weatherspoon 2000**] S. Weatherspoon, “Overview of IEEE 802.11b Security,” *Intel Technology Journal* (2nd Quarter 2000), http://download.intel.com/technology/itj/q22000/pdf/art_5.pdf
- [**Wei 2005**] W. Wei, B. Wang, C. Zhang, J. Kurose, D. Towsley, “Classification of Access Network Types: Ethernet, Wireless LAN, ADSL, Cable Modem or Dialup?,” *Proc. 2005 IEEE INFOCOM* (Apr. 2005).
- [**Wei 2006**] W. Wei, C. Zhang, H. Zang, J. Kurose, D. Towsley, “Inference and Evaluation of Split-Connection Approaches in Cellular Data Networks,” *Proc. Active and Passive Measurement Workshop* (Adelaide, Australia, Mar. 2006).
- [**Wei 2007**] D. X. Wei, C. Jin, S. H. Low, S. Hegde, “FAST TCP: Motivation, Architecture, Algorithms, Performance,” *IEEE/ACM Transactions on Networking* (2007).
- [**Weiser 1991**] M. Weiser, “The Computer for the Twenty-First Century,” *Scientific American* (Sept. 1991): 94–10. <http://www.ubiq.com/hypertext/weiser/SciAmDraft3.html>
- [**White 2011**] A. White, K. Snow, A. Matthews, F. Monrose, “Hookt on fon-iks: Phonotactic Reconstruction of Encrypted VoIP Conversations,” *IEEE Symposium on Security and Privacy*, Oakland, CA, 2011.
- [**Wigle.net 2012**] Wireless Geographic Logging Engine, <http://www.wigle.net>
- [**Williams 1993**] R. Williams, “A Painless Guide to CRC Error Detection Algorithms,” <http://www.ross.net/crc/crcpaper.html>
- [**Wilson 2011**] C. Wilson, H. Ballani, T. Karagiannis, A. Rowstron, “Better Never than Late: Meeting Deadlines in Datacenter Networks,” *Proc. 2011 ACM SIGCOMM*.
- [**WiMax Forum 2012**] WiMax Forum, <http://www.wimaxforum.org>
- [**Wireshark 2012**] Wireshark homepage, <http://www.wireshark.org>
- [**Wischik 2005**] D. Wischik, N. McKeown, “Part I: Buffer Sizes for Core Routers,” *ACM SIGCOMM Computer Communications Review*, Vol. 35, No. 3 (July 2005).
- [**Woo 1994**] T. Woo, R. Bindignavle, S. Su, S. Lam, “SNP: an interface for secure network programming,” *Proc. 1994 Summer USENIX* (Boston, MA, June 1994), pp. 45–58.
- [**Wood 2012**] L. Wood, “Lloyds Satellites Constellations,” <http://www.ee.surrey.ac.uk/Personal/L.Wood/constellations/iridium.html>

- [**Wu 2005**] J. Wu, Z. M. Mao, J. Rexford, J. Wang, “Finding a Needle in a Haystack: Pinpointing Significant BGP Routing Changes in an IP Network,” *Proc. USENIX NSDI* (2005).
- [**Xanadu 2012**] Xanadu Project homepage, <http://www.xanadu.com/>
- [**Xiao 2000**] X. Xiao, A. Hannan, B. Bailey, L. Ni, “Traffic Engineering with MPLS in the Internet,” *IEEE Network* (Mar./Apr. 2000).
- [**Xie 2008**] H. Xie, Y.R. Yang, A. Krishnamurthy, Y. Liu, A. Silberschatz, “P4P: Provider Portal for Applications,” *Proc. 2008 ACM SIGCOMM* (Seattle, WA, Aug. 2008).
- [**Yannuzzi 2005**] M. Yannuzzi, X. Masip-Bruin, O. Bonaventure, “Open Issues in Interdomain Routing: A Survey,” *IEEE Network Magazine* (Nov./Dec. 2005).
- [**Yavatkar 1994**] R. Yavatkar, N. Bhagwat, “Improving End-to-End Performance of TCP over Mobile Internetworks,” *Proc. Mobile 94 Workshop on Mobile Computing Systems and Applications* (Dec. 1994).
- [**YouTube 2009**] YouTube 2009, Google container data center tour, 2009.
- [**Yu 2006**] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman, “SybilGuard: Defending Against Sybil Attacks via Social Networks,” *Proc. 2006 ACM SIGCOMM* (Pisa, Italy, Sept. 2006).
- [**Zegura 1997**] E. Zegura, K. Calvert, M. Donahoo, “A Quantitative Comparison of Graph-based Models for Internet Topology,” *IEEE/ACM Transactions on Networking*, Vol. 5, No. 6, (Dec. 1997). See also <http://www.cc.gatech.edu/projects/gtim> for a software package that generates networks with a transit-stub structure.
- [**Zhang 1993**] L. Zhang, S. Deering, D. Estrin, S. Shenker, D. Zappala, “RSVP: A New Resource Reservation Protocol,” *IEEE Network Magazine*, Vol. 7, No. 9 (Sept. 1993), pp. 8–18.
- [**Zhang 2007**] L. Zhang, “A Retrospective View of NAT,” *The IETF Journal*, Vol. 3, Issue 2 (Oct. 2007).
- [**Zhang M 2010**] M. Zhang, W. John, C. Chen, “Architecture and Download Behavior of Xunlei: A Measurement-Based Study,” *Proc. 2010 Int. Conf. on Educational Technology and Computers (ICETC)*.
- [**Zhang X 2012**] X. Zhang, Y. Xu, Y. Liu, Z. Guo, Y. Wang, “Profiling Skype Video Calls: Rate Control and Video Quality,” *IEEE INFOCOM* (Mar. 2012).
- [**Zhao 2004**] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, J. Kubiatowicz, “Tapestry: A Resilient Global-scale Overlay for Service Deployment,” *IEEE Journal on Selected Areas in Communications*, Vol. 22, No. 1 (Jan. 2004).
- [**Zimmerman 1980**] H. Zimmerman, “OS1 Reference Model-The ISO Model of Architecture for Open Systems Interconnection,” *IEEE Transactions on Communications*, Vol. 28, No. 4 (Apr. 1980), pp. 425–432.
- [**Zimmermann 2012**] P. Zimmermann, “Why do you need PGP?” <http://www.pgpi.org/doc/whypgp/en/>
- [**Zink 2009**] M. Zink, K. Suh, Y. Gu, J. Kurose, “Characteristics of YouTube Network Traffic at a Campus Network - Measurements, Models, and Implications,” *Computer Networks*, Vol. 53, No. 4 (2009), pp. 501–514.

Index

A

- AAC (Advanced Audio Coding), 590
Abramson, Norman, 62, 454, 473
ABR ATM network service, 313
ABR (available bit-rate), 259
 spare available bandwidth advantage, 267
Abstract Syntax Notation One. *See* ASN.1
access control and SNMPv3, 777–778
access control lists, 734–736
access delay, 113
access ISP, 32–33
access networks, 12–18
 cable Internet access, 14–15, 460–461
 dial-up access, 16
 DSL (digital subscriber line), 13
 enterprises, 16–17
 Ethernet, 16–17
 FTTH (fiber to the home), 15
 home access, 13
 link-layer switches, 4
 LTE (Long-Term Evolution), 18, 533
 optical distribution network, 15
 satellite links, 16
2G, 3G, 4G (generation) wide-area
 wireless networks, 18, 547–554
wide-area wireless access, 18
 wireless LANs, 17
access points. *See* AP
access routers, 492
Accounting Management, 759, 764
acknowledgments, 210, 234–237
 piggybacked, 237
TCP (Transmission Control Protocol), 235–236, 244
 Telnet, 237–238
ACK (positive acknowledgments), 207, 240, 208, 212, 217, 235, 257
ACK receipt, 242
active optical networks. *See* AONs
active queue management algorithms. *See* AQM algorithms
adapters, 463–465
 MAC addresses, 471
 routers, 468
adaptive congestion control, 201
adaptive playout delay, 616–618
adaptive streaming, 600–601
adaptive HTTP streaming, 593
additive-increase, multiplicative-decrease.
 See also AIMD
address aggregation, 342
address indirection, 406
addressing
 Internet, 331–363
 processes, 90
Address Resolution Protocol. *See* ARP
Address Supporting Organization of ICANN, 345
ad hoc networks, 528
 802.15.1 networks, 544
 wireless hosts, 517
Adleman, Leonard, 684
Advanced Audio Coding. *See* AAC
Advanced Encryption Standard. *See* AES
Advanced Research Projects Agency. *See* ARPA
AES (Advanced Encryption Standard), 680
agent advertisement, 566–567
agent discovery, 565–567
agent solicitation, 567
aging time, 478
AH (Authentication Header) protocol, 720
AIMD (additive-increase, multiplicative-decrease) algorithm, 277–278, 280
Akamai, 114, 133, 609, 603–604, 35, 273

- alias hostname, 140
- ALOHAⁿ, 62–63, 454, 473
- ALOHA protocol, 62–63, 452, 473, 511, 453, 63, 453–455
- alternating-bit protocols, 214
- Amazon cloud, 608–610
- analog audio, 590
- anchor foreign agent, 563–564
- anchor MSC, 574
- Andreessen, Marc, 64
- anomaly-based IDSs (intrusion detection systems), 742
- anonymity, 738
- anycast, 356
- AONs (active optical networks), 15
- AP (access points), 517, 528–530, 538–540
- Apache Web server, 99, 156
- API (Application Programming Interface), 6, 89
- application architecture, 86–88
- application gateways, 732, 736–738
 - deep packet inspection, 739–740
- application-layer messages, 51, 54–55, 186
- application-layer protocols, 49–50
 - DNS (domain name system), 131, 132
 - electronic mail, 98
 - FTP (File Transfer Protocol), 51
 - HTTP (HyperText Transfer Protocol), 51, 97
 - Internet e-mail application, 97
 - proprietary, 97
 - public domain, 97
 - security, 705
 - SMTP (Simple Mail Transfer Protocol), 51, 97, 121
 - Telnet, 237
- Application Programming Interface. *See* API
- AQM (active queue management) algorithms, 329
- area border routers, 389
- ARPA (Advanced Research Projects Agency), 61, 511
- ARP (Address Resolution Protocol), 465–468, 498
- ARPAnet, 454, 473, 511
- ARP messages 467, 498
- ARP tables, 466–467, 481
- ARQ (Automatic Repeat reQuest) protocols, 207–208, 576–577
- ASN.1 (Abstract Syntax Notation One), 766, 770, 778–782
- ASN (autonomous system number), 394
- AS-PATH attribute, 394
- ASs (autonomous systems), 380–383
- association, 529–531
- assured forwarding PHB, 651
- asynchronous transfer mode. *See* ATM
- ATM ABR (available bit-rate) congestion control, 266–269, 313
- ATM (asynchronous transfer mode), 259, 512
 - complexity and cost, 470
 - services, 312–313
 - multiple service models, 312
 - Q2931b protocol, 654
- audio
 - AAC (Advanced Audio Coding), 590
 - glitches, 591
 - human speech, 590
 - MP3 (MPEG 1 layer 3), 590
 - PCM (pulse code modulation), 590
 - properties, 590–591
 - quantization, 590
 - removing jitter at receiver, 614–618
- authentication
 - cryptographic techniques, 675
 - end-point, 700–705
 - 802.11i, 729–730
 - MD5, 389
 - networks, 700–705
 - secret password, 701–703
 - SNMPv3, 777
 - WEP (Wired Equivalent Privacy), 726
 - 802.11 wireless LANs, 530
 - wireless station, 530
- authentication key, 692–693

authentication protocol examples, 700–705
authoritative DNS servers, 134–137, 499

- hostnames, 139
- IP addresses, 142
- names, 142

Automatic Repeat reQuest protocols. *See*
ARQ

autonomous system number. *See* ASN
autonomous systems. *See* ASs
available bit-rate. *See* ABR
average throughput, 44

B

backbone area, 389
bandwidth, 29, 281

- guaranteed minimal, 311
- link-level allocation, 639–640
- use-it-or-lose-it resource, 640
- video, 588–589, 594

 bandwidth flooding, 57
bandwidth provisioning, 635
bandwidth-sensitive applications, 92
Baran, Paul, 60
base HTML file, 99
base station controller. *See* BSC
base stations, 516–518, 528

- handoff between, 572–574

 base station system. *See* BSS
base transceiver station. *See* BTS
Basic Encoding Rules. *See* BER
basic service set. *See* BSS
beacon frames, 529–530
Bellman-Ford equation, 371–372
Bellovin, Steven M., 753–754
BER (Basic Encoding Rules), 780
BER (bit error rate), 520–521
Berners-Lee, Tim, 64
best-effort delivery service, 190
best-effort networks, 634–636
best-effort service, 190, 311–312,
612–614, 633–634
BGP (Border Gateway Protocol),
390–399, 498–499
ASN (autonomous system number), 394
attributes, 394, 395

BGP peers, 391
BGP sessions, 393
complexity, 391, 393
DV (distance-vector) algorithm, 374
eBGP (external BGP) session, 393–395
elimination rules for routes, 396
iBGP (internal BGP) session, 393–395
inter-AS routing protocols, 390–391,
393–399
peers, 391
prefix bits, 342, 344, 393
route advertisement, 391, 396–399
routes, 394–395
route selection, 395–396
routing policy, 397–399
routing tables, 399
session, 393
TCP connections, 391, 393
bidirectional data transfer, 205–206
binary exponential backoff algorithm,
457–458
BIND (Berkeley Internet Name Domain),
131
bit error rate. *See* BER
bit-level error detection and correction,
438–445
BITNET, 63
bits, 19
propagation delay, 37–38
BitTorrent, 86, 149–151

- accepting connections from other
hosts, 352
- chunks, 149
- developing, 182
- Kademlia DHT, 156
- P2P (peer-to-peer) protocol, 145, 182
- swarming data principles, 183
- trading algorithm, 150

 blades, 490
block ciphers, 678–681
Bluetooth, 518, 544–545
Boggs, David, 470, 473
Border Gateway Protocol. *See* BGP
border routers, 491
botnet, 56

- bottleneck link, 45, 279–280
 - broadcasting (link layer), 433, 464
 - channels, 433, 446
 - channel propagation delay, 456
 - frame, 467
 - links, 445, 475
 - broadcasting (network layer)
 - broadcast routing algorithms, 405
 - broadcast storm, 401
 - flooding, 401–403
 - N-way-unicast, 400–401
 - sequence number, 401
 - spanning-tree broadcast, 403–405
 - browsers, 101–102
 - client processes, 88
 - HTTP requests directed to Web cache, 110–111
 - BSC (base station controller), 550
 - BSS (base station system), 550
 - BSS (basic service set), 527–528
 - MAC address, 539
 - mobility among, 541–542
 - BTS (base transceiver station), 548–549
 - buffered distributors, 475
 - buffering
 - packets, 218
 - streaming video, 592
 - buffers
 - packet loss, 25
 - switch output interfaces, 476
 - TCP connection, 233
 - burst size, 646
 - bursty traffic, 60
 - bus, switching via, 326
- C**
- cable Internet access, 14–15
 - DOCSIS protocol, 52, 460
 - link-layer protocols, 460–461
 - cable modems, 15, 20, 460–461
 - cable modem termination system. *See* CMTS
 - CA (Certification Authority), 697–699
 - certifying public keys, 708
 - call admission, 653–654
 - call setup protocol, 654
 - canonical hostnames, 131–132, 140
 - care-of-address. *See* COA
 - carrier sense multiple access protocol. *See* CSMA protocol
 - carrier sense multiple access with collision detection. *See* CSMA/CD
 - carrier sensing, 454–456
 - CBC (cipher-block chaining), 681–682
 - CBR (constant bit rate) ATM network service, 312
 - CDMA (code division multiple access), 522–526
 - CDNs (Content Distribution Networks), 588, 603–608
 - access networks of ISPs (Internet Service Providers), 603–604
 - cluster selection strategy, 606–608
 - data centers, 604
 - delay and loss performance measurements, 606
 - DNS intercept/redirect, 604–605
 - IP anycast, 607–608
 - Netflix, 608
 - operation, 604–605
 - replicating content across clusters, 604
 - cell phones
 - growth of, 513
 - Internet access, 65, 546–554
 - cellular networks
 - architecture, 547–550
 - cells, 548–549
 - handoffs in GSM, 572–574
 - managing mobility, 570–575
 - routing calls to cellular user, 571–572
 - 2G, 3G, 4G networks, 547–554
 - Cerf, Vinton G., 62, 231, 431–432, 511
 - CERT Coordination Center, 674
 - certificates, 698–699, 713–714
 - Certification Authority. *See* CA
 - channel numbers, 529
 - channel partitioning protocols, 447
 - CDMA (code division multiple access) protocol, 449, 522–526

- FDM (frequency-division multiplexing), 448
- TDM (time-division multiplexing), 448
- channels
 - with bit errors and reliable data transfer, 207–212
 - 802.11 wireless LANs, 529–531
 - losing packets, 212–215
 - propagation delay, 456
- checksums, 203, 442
 - ACK/NAK packets, 210
 - calculation, 203
 - Internet, 203
 - poor cryptographic hash function, 690
- TCP (Transmission Control Protocol), 334
 - UDP (User Datagram Protocol), 202–204, 334
- chipping rate, 522
- choke packet, 266
- chosen-plaintext attack, 678
- chunks, 149
 - delaying playout, 615–618
- CI (congestion indication) bit, 268–269
- CIDR (Classless Interdomain Routing), 342, 344, 496
 - cipher-block chaining. *See* CBC
 - ciphertext, 675, 676
 - ciphertext-only attack, 677
- circuit-switched networks
 - end-to-end connection, 28
 - multiplexing, 28–30
 - versus packet switching, 30–31
 - reserving time slots, 30–31
 - sending packets, 28
 - telephone networks, 27
- circuit-switched routing algorithm, 379
- Cisco Systems, 65, 323, 732, 740
 - routers and switches, 325, 326
 - dominating network core, 323
- Clark, Dave, 303, 585
- classful addressing, 344
- Classless Interdomain Routing. *See* CIDR
- cleartext, 675
- Clear to Send control frame. *See* CTS control frame
- client application buffer, 597–598
- client buffering, 594–595
- clients, 10, 86, 88, 89, 156
 - initiating contact with server, 163
 - HTTP, 198
 - IP addresses, 162
 - matching received replies with sent queries, 140
 - port number, 162
 - prefetching video, 597
 - receiving and processing packets from, 162
 - TCP socket creation, 163
 - Web caches as, 111
- client-server application
 - developing, 157–168
 - TCP, 157
 - UDP, 157
 - well-known port number, 157
- client side socket interface, 99
- client socket, 160, 163, 165
- cloud applications and data center networking, 490–495
- cluster selection strategy, 606–608
- CMTS (cable modem termination system), 15, 460
- CNAME record, 140
- COA (care-of-address), 559, 565
- coaxial cable, 20, 474
- code division multiple access. *See* CDMA
- collision detection, 454–455
- collisions, switch elimination, 479
- Comcast, 601, 763–764
- commercial ISPs (Internet Service Providers), 64
- communication satellite, 21
- computer networking
 - history of, 60–66
- computer networks, 2
- conditional GET, 114–116
- confidentiality, 672–673, 706–707, 718, 720, 724
- Configuration Management, 759, 764

- congestion
 - causes and costs, 259–265
 - dropping packets, 265
 - large queuing delays, 261
 - transmission capacity wasted, 265
 - unneeded retransmissions by sender, 263
- congestion avoidance, 274–276
- congestion control, 190, 240, 250, 596
 - ABR (available bit-rate) service, 259
 - AIMD (additive-increase, multiplicative-decrease), 277–278
 - approaches, 265–266
 - ATM (asynchronous transfer mode) networks, 259
 - end-to-end, 266, 269
 - network-assisted, 266, 269
 - principles, 259–269
 - source quench message, 353
 - TCP (Transmission Control Protocol), 269–272, 274–283
 - UDP (User Datagram Protocol), 282
- congestion indication bit. *See* CI bit
- congestion window, 269–270, 272–277, 282
- connection-establishment request, 195
- connectionless service, 313
- connectionless transport and UDP (User Datagram Protocol), 198–204
- connection-oriented service, 94, 313–314
- connection-oriented transport and TCP (Transmission Control Protocol), 230–258
- connection replay attack, 717
- connections, persistent and non-persistent, 100–103
- connection setup, 310
- connection sockets and processes, 198
- connection state, 231, 315
- constant bit rate ATM network service.
 - See* CBR ATM network service
- container-based MDCs (modular data centers), 494
- Content Distribution Networks. *See* CDNs
- content provider networks, 34–35
- continuous playout, 591–592
- control connection, 117
- controlled flooding, 401–403
- control plane software, 331
- conversational applications, 592–593
- cookies, 108–110
- correspondent, 557, 561, 563
- corrupted ACKs or NAKs, 209–210
- countdown timer, 214
- coverage area, 515
- CRC (cyclic redundancy check) codes, 443–445
- crossbar switch, 326
- cryptographic hash functions, 689–691
- cryptographic techniques, 675
- cryptography, 675–688
 - ciphertext, 675–676
 - cleartext, 675
 - confidentiality, 675
 - cryptographic hash functions, 689–691
 - decryption algorithm, 676
 - encryption algorithm, 675–676
 - keys, 676
 - plaintext, 675–676
 - public key encryption, 683–688
 - symmetric key cryptography, 676–682
- CSMA (carrier sense multiple access), 453–456
 - carrier sensing, 455–456
 - collisions, 455–456
- CSMA/CA (carrier sense multiple access with collision avoidance), 526, 531–537
- CSMA/CD (carrier sense multiple access with collision detection), 456–459
 - collision detection, 456–459
 - efficiency, 458–459
 - Ethernet, 475
- CSNET (computer science network), 63
- CTS (Clear to Send) control frame, 535–537
- cumulative acknowledgments, 222, 236, 243

customer ISPs, 34
 customer-provider relationship, 33
 cyclic redundancy check codes. *See* CRC codes

D

DARPA (US Department of Defense Advanced Research Projects Agency), 62, 431
 DASH (Dynamic Adaptive Streaming over HTTP), 600–601
 databases, implementing in P2P network, 151–156
 data center networks, 490–495
 data centers, 86
 border routers, 491
 CDNs (Content Distribution Networks), 604
 container-based MDCs (modular data centers), 494
 costs, 490
 data center networks, 490–495
 hierarchy of routers and switches, 492–493
 hosts, 490
 interconnection architectures and network protocols, 493
 Internet services, 86
 load balancing, 491–492
 servers, 11
 data definition language, 765
 data encryption algorithm and WEP (Wired Equivalent Privacy), 726–728
 Data Encryption Standard. *See* DES
 DATA frame, 535
 datagram networks, 313, 317–320
 datagrams, 55, 189
 detecting bit errors, 334
 extending IP header, 335
 fragmentation, 335–338
 Internet Protocol (IP) v4 format, 332–338
 labeling, 487
 length, 334
 moving between hosts, 51–52
 passed to transport layer, 337

reassembling in end systems, 336
 sending off subnet, 468–469
 source and destination IP addresses, 334–336
 TOS (type of service) bits, 333
 transport-layer segments, 242
 TTL (time-to-live) field, 334
 datagram service and network layer, 364
 data integrity, 363, 712
 data link layer, 50, 53
 Data-Over-Cable Service Interface Specifications. *See* DOCSIS
 data plane and hardware, 331
 DATA SMTP command, 123, 124
 data transfer, 713
 bidirectional, 205–206
 reliable, 91, 204–230
 SSL (Secure Sockets Layer), 714–715
 unidirectional, 205
 unreliable, 206
 VC (virtual-circuit) networks, 316
 DDoS (distributed denial-of-service) attacks, 56–58, 143–144
 DECnet architecture, 266
 decentralized routing algorithm, 366
 decryption algorithm, 676, 283
 deep packet inspection, 739–741
 Deering, Steve, 584
 default name server, 136
 default router, 364
 delaying playout, 615–618
 delays
 comparing transmission and propagation delays, 38–39
 end systems, 43–44
 end-to-end, 42–44
 nodal processing, 36
 packetization, 44
 packets, 36
 packet-switched networks, 35–39
 processing, 36–37
 propagation, 36–37
 queuing, 36–37, 39–42
 total nodal, 36
 transmission, 36–37

- delay-sensitive, 592
demilitarized zone. See DMZ
 demultiplexing, 191–198
 connectionless, 193–194
 connection-oriented, 194–197
 denial-of-service attacks. *See DoS attacks*
 DES (Data Encryption Standard), 680
 destination port numbers, 192, 234
 destination router, 364
 DHCP (Dynamic Host Configuration Protocol), 345–349
 IP addresses dynamically assigned, 630
 DHCP request message, 348, 495–496
 DHCP servers, 346–347, 350, 495, 497
 DHTs (distributed hash tables), 145, 151–156
 dial-up access, 16
 DIAMETER protocol, 530, 730
 differentiated service, 634
 Diffie-Hellman algorithm, 683, 687, 725
 Diffserv, 648–652
 DIFS (Distributed Inter-frame Space), 533
 digital audio, 590
 digital signatures, 707
 compared with MAC (message authentication code), 696–697
 hash functions, 695–696
 message integrity, 695
 overheads of encryption and decryption, 695
 PGP (Pretty Good Privacy), 710
 public key certification, 697–699
 public-key cryptography, 693–694
 digital subscriber line. *See DSL*
 digital subscriber line access multiplexer. *See DSLAM*
 Dijkstra's least-cost path algorithm, 367–368, 388, 390
 OSPF (Open-Shorest Path First), 388
 dimensioning best-effort networks, 634–636
 directory service, 97
 direct routing, 563–564
 Direct Sequence Wideband CDMA. *See DS-WCDMA*
 distance vector, 372
 distance-vector algorithm. *See DV algorithm*
 Distance-Vector Multicast Routing Protocol. *See DVMRP*
 distributed applications, 5–6
 distributed attacks. *See DDoS attacks*
 Distributed Inter-frame Space. *See DIFS*
 DMZ (demilitarized zone), 741
 DNS
 DNS database, 142, 144
 DNS (domain name system), 51, 63, 98, 130–144, 497
 additional delay to Internet applications, 131
 application-layer protocols, 131–132
 attacks and vulnerabilities, 143–144
 caching, 138–139
 CDN requests, 604–605
 client-server paradigm, 132
 DDoS attack against targeted host, 143–144
 distributed and hierarchical database, 131, 134–138
 host aliasing, 131–132
 hostname-to-IP-address translation service, 133–139
 improving delay performance, 138
 iterative queries, 137–138
 load distribution, 132–133
 obtaining presence in, 392
 operational overview, 133–139
 querying and replaying messages, 133
 query messages, 136, 140–142, 497
 recursive queries, 137–138
 reply messages, 136, 140–142, 499
 rotation, 132–133
 RRs (resource records), 139–140, 498–499
 secure communication, 674
 servers, 134
 services provided by, 131–133

- translating hostnames to IP addresses, 133–139
- underlying end-to-end transport protocol, 132
- UPDATE option, 144
- DNS servers, 131–135, 495
- authoritative DNS servers, 134, 135–136, 140
 - BIND (Berkeley Internet Name Domain), 131
 - DDoS bandwidth-flooding attack, 143
 - discarding cached information, 139
 - distant centralized database, 134
 - hierarchy, 136
 - local DNS server, 136–137
 - recursion, 140
 - root DNS servers, 134, 135
 - single point of failure, 133
 - TLD (top-level domain) DNS servers, 134
- DOCSIS (Data-Over-Cable Service Interface Specifications), 15, 52, 460
- domain name system. *See* DNS
- DoS (denial-of-service) attacks, 57, 735, 740
- fragmentation and, 338
 - SYN flood attack, 253, 257
- dotted-decimal notation, 339
- dropping packets, 265
- drop-tail, 329
- DSLAM (digital subscriber line access multiplexer), 13–14
- DSL (digital subscriber line), 13–14, 20
- DS-WCDMA (Direct Sequence Wideband CDMA), 552
- dual-stack approach, 359
- duplicate ACKs, 211, 247–248
- duplicate data packets, 210, 213–214
- DV (distance-vector) algorithm, 366, 371–380
- DVMRP (Distance-Vector Multicast Routing Protocol), 411–412
- Dynamic Adaptive Streaming over HTTP. *See* DASH
- Dynamic Host Configuration Protocol. *See* DHCP
- dynamic routing algorithm, 366
- ## E
- EAP (Extensible Authentication Protocol), 729–730
- eavesdropping, 673
- eBGP (external BGP), 393–395, 397
- EC2, 66
- edge router, 12
- EFCI (explicit forward congestion indication) bit, 268
- 802.15.1, 544–545
- 802.11i, 728–731
- 802.11n wireless network, 527
- 802.1Q frames, 484–486
- 802.11 wireless LANs, 17, 515, 518, 526–546, 548
- address fields, 538–540
 - APs (access points), 517, 528
 - architecture, 527–531
 - association, 529–531
 - authentication, 530, 728–731
 - base station, 528
 - BSS (basic service set), 527–528
 - channels, 529–531
 - collision detection, 532
 - CSMA/CA (CSMA with collision avoidance), 526, 531–537
 - data rates, 526–527
 - frequency band, 526
 - link-layer frames, 526
 - MAC addresses, 528
 - MAC protocols, 531–537
 - reducing transmission rate, 526
 - transmitting frames, 532
- elastic applications, 92
- electromagnetic noise, 519
- e-mail, 64, 86, 97, 118–130
- access protocols, 125–126
 - application-layer protocols, 98
 - security, 705–711
 - Web-based e-mail system, 120
- encapsulation, 53–55, 565–567

- encapsulation/decapsulation and mobile IP, 565
- Encapsulation Security Payload protocol. *See* ESP
- encryption, 712
- cryptography, 675–688
 - SNMPv3, 777
- end-point authentication, 59, 673
- end systems, 2, 4–6, 10
 - API (Application Programming Interface), 6
 - applications running on, 6
 - communication links, 4
 - connecting, 5
 - datagram reassembly, 336
 - delays, 43–44
 - hosts, 10
 - IP addresses, 26
 - processes, 84, 88
 - TCP protocol, 231
 - transport-layer protocols, 188
 - end-to-end congestion control, 266
 - end-to-end delay, 42–43, 613
 - Diffserv, 652
 - Traceroute program, 42–43
 - end-to-end principle, 203
 - end-to-end throughput, 44–47
 - enterprises and access networks, 16–17
 - EPC (Evolved Packet Core), 553
 - ER (explicit rate) field, 269
 - error checking
 - link-layer protocols, 203
 - parity checks, 440–442
 - transport layer, 203
 - error concealment, 620–621
 - error detection
 - ARQ (Automatic Repeat reQuest) protocols, 208
 - checksumming methods, 203, 442–443
 - CRC (cyclic redundancy check) codes, 443–445
 - parity bits, 440–442
 - two-dimensional parity scheme, 441–442
 - ESP (Encapsulation Security Payload), 720, 723
 - ESTABLISHED state, 254
 - Estrin, Deborah, 303, 584–585
 - Ethernet, 16–17, 52, 63, 437, 445, 469–476
 - binary exponential backoff algorithm, 457–458
 - broadcast link, 475
 - carrier-sensing random access, 531–532
 - collision-detection algorithm, 532
 - CSMA/CD protocol, 474–475
 - frame format, 474
 - home networks, 17
 - link-layer and physical-layer specification, 474
 - local area networking, 470
 - MSS (maximum segment size), 233
 - MTU (maximum transmission unit), 471
 - multiplexing network-layer protocols, 471–472
 - physical-layer protocols, 52, 473–474
 - standardized, 474
 - store-and-forward packet switching, 475
 - switch-based star topology, 475
 - switches, 17, 470, 475, 496
 - 10BASE-2, 473–474
 - 10BASE-T, 473–474
 - 10GBASE-T, 474–475
 - EWMA (exponential weighted moving average), 240
 - expedited forwarding PHB, 651
 - explicit forward congestion indication bit. *See* EFCI
 - explicit rate field. *See* ER (explicit rate) field
 - exponential weighted moving average. *See* EWMA
 - extended FSM (finite-state machine), 220, 223
 - Extensible Authentication Protocol. *See* EAP
 - external BGP session. *See* eBGP session

F

fairness

- AIMD algorithm, 280
- congestion-control mechanism, 280
- parallel TCP connections, 282
- TCP (Transmission Control Protocol), 279–282

- TDM (time-division multiplexing), 448
- UDP (User Datagram Protocol), 282

Fault Management, 758–759, 764

FCFS (first-come-first-served) scheduling, 329, 641

FDDI (fiber distributed data interface), 460, 470

FDM (frequency-division multiplexing), 28–30, 448, 549

FDM/TDM systems, 549–550

FEC (forward error correction), 442, 613, 618–619

FHSS (frequency-hopping spread spectrum), 544

fiber distributed data interface protocol. *See* FDDI

fiber optics, 20–21

- 100 Mbps Ethernet, 475

fiber to the home. *See* FTTH

FIFO (first-in-first-out), 637–638, 641–642

file sharing, 86

file transfer, 97, 116–118

filtering, 476–477

FIN bit, 235, 254

finite-state machine. *See* FSM

FIOS service, 15–16

firewalls, 673, 730–738

- application gateways, 732, 736–738

- authorized traffic, 731

- blocking packets, 355, 596

- connection table, 732–736

- filtering, 733–736

- malicious packet attacks, 355

first-come-first-served scheduling.

See FCFS

first-in-first-out order. *See* FIFO

fixed-length labels, 487

fixed playout delay, 615

flag field, 235, 334–336

flooding, 401–405

flow control, 240

- different from congestion control, 220

TCP, 250

flow (of packets), 311, 357

foreign address, 559

foreign agent, 557, 566

COA (care-of-address), 567

mobile IP, 565

receiving and decapsulating datagram,

561, 562

registration with home agent, 562, 568–569

foreign networks, 557–559

forwarding, 305, 308–310, 321–322, 322, 476–477, 649

forwarding function, 320–321

forwarding plane, 321

forwarding tables, 26–27, 308–309, 317–318, 322–323

adding entries, 396–397

configuring, 308–309, 364

datagram networks, 319

directing datagrams to foreign network, 558

routing algorithms, 364

VC networks, 319

4G systems, 18, 553–554

fourth-generation of wide-area wireless networks. *See* 4G

fragmentation of datagrams, 334–338

frames, 52, 433–434, 436

frequency-division multiplexing. *See* FDM

frequency-hopping spread spectrum. *See* FHSS

FSM (finite-state machine), 206–207

extended, 220, 223

initial state, 207

FTP (File Transfer Protocol), 51, 86,

97–98, 116–118

FTTH (fiber to the home), 15–16

full-duplex service, 232

full-table block ciphers, 679–680

fully connected topology, 493–494

G

Gateway GPRS Support Nodes. *See* GGSNs
 gateway routers, 380–381
 import policy, 395
 packet filtering, 732
 prefixes, 393
 GBN (Go-Back-N) protocol, 218–224
 Generalized Packet Radio Service. *See* GPRS
 generator, 443
 geographically closest, 606
 geostationary satellites, 21
 GET method, 104–105, 115
 GGSNs (Gateway GPRS Support Nodes), 552
 Gigabit Ethernet, 475
 global routing algorithm, 365–366
 Global System for Mobile Communications standards. *See* GSM
 global transit ISPs (Internet Service Providers), 32–33
 GMSC (Gateway Mobile services Switching Center), 570
 Go-Back-N protocol. *See* GBN protocol
 Google, 34–25, 64–66, 86, 114, 431, 603, 610
 GPRS (Generalized Packet Radio Service), 552
 graphs, 364–365
 ground stations, 21
 GSM (Global System for Mobile Communications), 547, 549–550, 570
 anchor MSC, 574
 BTS (base transceiver station), 548–549
 cells, 548–549
 encoding audio, 628
 handoffs, 572–574
 home network, 570
 home PLMN (home public land mobile network), 570
 indirect routing, 570

mobility, 576

routing cells to mobile user, 571–572
 visited network, 570
 guaranteed delivery, 311
 guided media, 19

H

handoff, 517–518
 GSM, 572–574
 handshake, 94, 231–232, 252–253, 713, 716
 hard guarantee, 634
 hardware data plane, 331
 hardware-implemented protocols, 9
 hash functions, 152, 707, 714
 digital signatures, 695–696
 HDLC (high-level data link control), 445
 header checksum, 334
 header fields, 55
 head-of-line blocking. *See* HOL blocking
 HFC (hybrid fiber coax), 14
 hidden terminals, 521, 534–537
 hierarchical routing, 379–383
 higher-layer protocols and wireless networks and mobility, 575–577
 high-level data link control. *See* HDLC
 HLR (home location register), 570, 572
 HMAC standard, 692–693
 HOL (head-of-the-line) blocking, 330–331
 home agents, 557–562, 565–566, 568–569
 home location register. *See* HLR
 home MSC, 574
 home networks, 13, 557
 Ethernet, 17
 GMSC (Gateway Mobile services Switching Center), 570
 GSM (Global System for Mobile Communications), 570
 HLR (home location register), 570
 home agent, 558–559
 IP addresses, 349–352
 mobile nodes, 559
 PLMN (home public land mobile network), 570
 WiFi, 17

hostnames, 130
 alias, 132
 canonical, 131
 translating to IP addresses, 131, 132, 133–139

hosts, 2, 4, 10–11
 aliasing, 131–132, 140
 ARP table, 466
 assigning IP addresses to, 345–349
 canonical hostname, 140
 changing base station, 517–518
 connected into network, 338
 data centers, 490
 default router, 364
 DHCP discovery message, 530
 dial-up modem connection, 486
 hostnames, 130–131
 interfaces, 338
 IP addresses, 90, 130–131, 190, 465
 link-layer addresses, 462–463
 load balancer, 491–492
 local DNS server, 136
 MAC addresses, 465
 monitoring, 756
 moving datagrams between, 51–52
 network-layer addresses, 462, 465
 network-layer protocols, 471–472
 process-to-process delivery service, 191–198
 storing routing information, 380
 wireless, 514

hot-potato routing, 382

HSP (High Speed Packet Access), 552

HTML, 64, 100–101

HTTP (HyperText Transfer Protocol), 51, 97–100, 127
 client program, 98
 conditional GET, 114–116
 cookies, 108–110
 GET request, 103–105, 499–500, 596, 601, 610
 If-Modified-Since: header line, 115
 message format, 98, 103–108

non-persistent connections, 100–103
 persistent connections, 103, 124
 pull protocol, 124
 POST method, 104–105
 PUT method, 105
 response message, 105–108, 124, 500, 596
 server program, 98, 100
 SMTP comparison with, 124
 stateless, 108
 SSL security, 712
 stateless protocols, 100, 117
 TCP and, 99–100, 116, 200
 Web caching, 110–114

HTTP streaming, 593, 597–600, 611
 prefetching video, 596–597

hubs, 470

Hulu, streaming stored video, 591

human protocols, 7–8

human speech, 590

hybrid fiber coax. *See* HFC

hypertext, 64

HyperText Transfer Protocol. *See* HTTP

I

iBGP (internal BGP), 393–395, 397

IBM SNA architecture, 62, 266

ICANN (Internet Corporation for Assigned Names and Numbers), 142, 345

ICMP (Internet Control Message Protocol), 306, 353–355, 359
 datagram, 258
 messages, 353–354
 IPv6, 359

IDEA, 710

identification field (IP), 336

IDSs (intrusion detection systems), 355, 673, 739–742

IEEE 802.3 CSMA/CD (Ethernet) working group, 474

IEEE 802 LAN/MAN Standards Committee, 5

IEEE managing MAC address space, 464

- IETF (Internet Engineering Task Force), 5, 668
 Address Lifetime Expectations working group, 356
 standards for CAs, 699
 If-Modified-Since: header line, 115
 IGMP (Internet Group Management Protocol), 359, 407–409
 IKE (Internet Key Exchange), 725
 IMAP (Internet Mail Access Protocol), 127, 129
 IMAP server, 129
 import policy, 395
 in-band, 117
 indirect routing,
 GSM (Global System for Mobile Communications), 570
 mobile IP standard, 562
 mobile node, 559–562
 triangle routing problem, 563
 infrastructure wireless networks, 517, 528
 in-order packet delivery, 311
 input ports, 320
 packet queues, 327–331
 processing, 322–324
 switching to output ports, 324–326
 input processing, 322–324
 instantaneous throughput, 44
 instant messaging, 65, 83, 632
 Intel 8254x controller, 437
 interactivity, 591
 inter-AS routing, 390–391, 393–399
 inter-AS routing protocols, 382, 398
 BGP (Border Gateway Protocol), 390–391, 393–399
 interconnection network, 326
 interfaces
 assigning IP addresses to, 345–349
 IP addresses, 338–339
 routers, 468
 interior gateway protocols, 384
 interleaving, 618–620
 Intermediate-System-to-Intermediate-System routing algorithm. *See* IS-IS routing algorithm
 internal BGP session. *See* iBGP session
 International Organization for Standardization. *See* ISO
 Internet
 access networks, 12–18
 address assignment strategy, 342
 addressing, 331–363
 best-effort service model, 190, 311–313, 612–614, 636
 cellular access, 546–554
 commercialization, 64
 connecting end systems, 5
 connectivity, 392
 delivering data, 6
 difficulty making changes to, 303
 distributed applications, 5–6
 e-mail, 64, 118–130, 705–711
 end systems, 2, 4, 10
 flag day, 359
 forwarding, 331–363
 global traffic, 4
 history of, 60–66
 hosts, 2, 4
 inter-AS routing, 390–391, 393–399
 inter-domain protocol, 498
 intra-AS routing, 384–390
 ISPs (Internet Service Providers), 4, 32–35
 IXPs (Internet exchange points), 34
 layered architecture, 47–53
 link-layer overview, 434–438
 link-layer switches, 4, 476–482
 mobile devices, 550–552
 multicast routing, 411–412
 network layer, 332
 network of networks, 3–5, 32–35
 obtaining presence on, 392–393
 packet forwarding, 26–27
 protocols, 5, 9
 protocol stack, 50
 root DNS servers, 135
 routers, 4
 routing protocols, 27, 383–399
 service classes, 636–640
 service provider private networks, 66
 services description, 5–7
 standards, 5

- 3G and 4G cellular networks, 65, 547–554
- TLD (top-level domain) servers, 135
- transport layer overview, 189–191
- transport services, 93–96
- Web, 64, 98–100, 111
- wireless access, 17–18
- Internet API, 6, 156–158
- Internet applications, 83, 100
- application-layer protocols, 96–97
- Internet checksum, 203, 334, 442
- Internet commerce, 64, 86
- cloud, 66
- Internet Control Message Protocol. *See* ICMP
- Internet Corporation for Assigned Names and Numbers. *See* ICANN
- Internet Engineering Task Force. *See* IETF
- Internet exchange points. *See* IXPs
- Internet Group Management Protocol. *See* IGMP
- Internet hosts. *See* hosts
- Internet Key Exchange. *See* IKE
- Internet Mail Access Protocol. *See* IMAP
- Internet phone application, 612–614, 618–621
- Internet Protocol. *See* IP
- Internet registrar, 392
- Internet routing protocols, 353
- Internet Service Providers. *See* ISPs
- Internet-Standard Management Framework, 764–778
- Internet telephony, 87, 612–623
- Skype, 621–623
- internetting, 62
- Internet transport protocols, 95
- intra-AS (autonomous system) routing protocols, 380, 381, 397–398
- intra-AS routing, 388–390
- intra-domain routing and DNS servers, 498–499
- intrusion detection, 758
- intrusion prevention system. *See* IPS
- intrusion detection systems. *See* IDSs
- IP addresses, 26, 90, 130–131
- adapters, 465
- address aggregation, 342
- allocating, 345
- authoritative DNS server, 142
- block, 345
- CIDR (Classless Interdomain Routing), 342, 344
- Classful addressing, 344
- destination, 334–335
- distinguishing among devices, 344
- DNS, 130–144
- dotted-decimal notation, 339
 - globally unique, 339
 - hierarchical structure, 130, 464
 - home networks, 349–352
 - increasing size of, 356
 - interfaces, 338–339
 - IP broadcast, 344
 - lease time, 347
 - local DNS server, 136
 - mobility, 556–559
 - obtaining, 495
 - prefix, 342, 344
 - private addresses, 349–350
 - range, 392
 - routers, 465, 468
 - setting up call to known, 627–629
 - source, 334–335
 - subnets, 340, 342, 344
 - translating hostnames to, 131–139
- IP anycast, 607–608
- IP broadcast address 344, 347
- IP datagrams, 233, 496
- attacks and, 355
 - encryption of payloads, 363
 - Ethernet frames, 471
 - fragmentation, 335–338
 - MTU (maximum transmission unit), 335
 - security, 718
 - transport-layer protocols, 334
- IP header, 202, 335
- IP-in-IP encapsulation, 567
- IP (Internet Protocol), 5, 51–52, 190, 306, 331–332, 353, 387
- best-effort delivery service, 190, 311–312, 612–614, 633–634
- confining Internet's development, 512

- IP (Internet Protocol) (*continued*)
 datagrams, 242, 332–338
 de facto standard for internetworking, 512
 Internet addressing and forwarding, 331–363
 multicasting, 593
 network layer, 305–307, 332
 pervasiveness, 431
 routers, 53
 security, 362–363
 separation from TCP, 62
 IP multicast, 412
 IPng (Next Generation IP), 356
 IP protocol version 4. *See* IPv4
 IP protocol version 6. *See* IPv6
 IPsec, 362, 72–724
 IPS (intrusion prevention system), 355, 740
 IP spoofing, 59–60, 701
 IP subnet, 541–542
 IP tunnels, 303
 IPTV, 87
 IPv4 addresses, 334, 356
 IPv6 addresses, 356, 512
 IPv4 addressing, 338–352
 IPv4 datagrams, 332–335
 format, 306
 IPv6 datagrams, 356–359
 IPv4 header, 636–637
 IPv6 header, 359
 IPv4 (IP protocol version 4), 160, 165,
 331, 336
 IPsec, 362
 transitioning to IPv6, 359–362
 IPv6 (IP protocol version 6), 306, 332,
 356–362
 dual-stack approach, 360
 flow labeling and priority, 357
 IP addresses, 351, 356
 transitioning from IPv4, 359–362
 tunneling, 360–361
 IPv6/IPv4 nodes, 359–360
 IS-IS (Intermediate-System-to-
 Intermediate-System), 405, 498
 IS-IS protocol, 384
 ISO (International Organization for
 Standardization), 52, 758, 770
 ISPs (Internet Service Providers), 32–35
 access networks, 12–18
 ASs (autonomous systems), 383
 customer-provider relationship, 33
 end-to-end service, 651–652
 global transit, 32
 low-tier and upper-tier, 5
 multi-homing, 33–34
 naming and addressing conventions, 5
 obtaining set of addresses from, 345
 peer, 34
 peering agreements, 399
 iterative queries, 137–138
 ITU-T (International Telecommunication
 Union), 699, 770
 IXPs (Internet exchange points), 33, 34
- J**
 Jacobson, Van, 302, 303, 584
 Java and client-server programming, 157
 jitter, 614–618
- K**
 Kademlia DHT, 156
 Kahn, Robert, 62, 231, 431–432, 511
 KanKan, 87, 588, 611–612
 key derivation, 713–714
 key management, 726
 Kleinrock, Leonard, 60–62, 80–82, 431, 511
 known-plaintext attack, 677
- L**
 label-switched routers, 488
 Lam, Simon S., 511–512
 LANs (local area networks), 16
 broadcast address, 464
 configured hierarchically, 482–483
 Ethernet, 470
 hub-based star topology, 470
 MAC address, 463
 switches, 17, 470, 475, 483, 496
 VLANs (virtual local area networks),
 482–486

- Last-Modified: header line, 106, 115
 layered architecture, 47–48, 51–53
 layer-2 packet switch, 480
 layer-4 switch, 492
LDNS (Local DNS Server), 136–137, 605–606
 leaky bucket mechanism, 646–648
 least-cost path, 365, 367–368
 - Bellman-Ford equation, 371–372**LEO (low-earth orbiting) satellites**, 21–22
Limelight, 35, 114, 604, 609
 limited-scope flooding, 405
 link-cost changes and DV (distance-vector) algorithm, 376–377
 link-layer acknowledgments, 532
 link-layer addressing, 462–469
 link-layer frames, 55, 434, 436, 438–445, 461–462
 link-layer protocols, 52
 - cable Internet access, 460–461
 - error detection and correction, 437
 - services 436–437**link layer**, 52, 433–436
 - bit-level error detection and correction, 438–445
 - broadcast channels, 433
 - CRC (cyclic redundancy check) codes, 443–445
 - error-detection and-correction techniques, 438–445
 - IEEE protocols, 444
 - implementation, 437–438
 - link-layer frame, 434
 - multiple access protocols, 445–461
 - networks as, 486–490
 - point-to-point communication link, 434
 - services, 436–437
 - switches, 461
 - wireless links, 17–18, 519–522**link-layer switches**, 4, 22, 53–54, 310, 476–482
 link rates, 515
 links, 434
 - broadcast, 445
 - heterogeneous, 479
 MPLS header format, 488
 point-to-point, 445
 transmission rates, 4
link-scheduling disciplines
 - FIFO (first-in-first-out), 641–642
 - priority queuing, 642–643
 - round robin queuing discipline, 643–644
 - WFQ (weighted fair queuing), 644–645
 - work-conserving round robin discipline, 644**link-state advertisements**. *See* LSAs
link-state broadcast, 366–367
link-state messages, 689
link-state protocols, 388, 400
link-state routing algorithms, 366–371
link virtualization, 486–490
link weights, 390
load balancer, 491–492
load distribution, 132–133
load-insensitive routing algorithm, 366
local area networks. *See* LANs
Local DNS Server. *See* LDNS
local ISP, 392
logical communication between processes, 186
longest prefix matching rule, 318–319
Long-Term Evolution. *See* LTE
loss recovery schemes, 618
loss-tolerant, 91, 592–593
lost packets, 262–263
low-earth orbiting satellites. *See* LEO
 satellites
LSAs (link-state advertisements), 405
LTE (Long-Term Evolution), 18, 553–554
- ## M
- MAC addresses**, 463–465, 497
 adapters, 464–465, 465, 471
 AP (access point), 528
 BSS, 539
 802.11 wireless LAN, 528
 flat structure, 464
 no two adapters have same, 464
 permanent, 463–464
 switches, 477, 480

- MAC-based VLANs (virtual local area networks), 486
- MAC (message authentication code), 691–693, 777
 - compared with digital signatures, 696–697
- MAC (multiple access protocols), 436, 464, 531
 - 802.11 wireless LANs, 531–537
- mail access protocols, 125–130
- mailbox, 120–121
- mail clients, 97, 125
- mail servers, 97, 119–121, 125–126
- main-in-the-middle attack and DNS (domain name system), 143
- malicious packet attacks, 355
- malware, 56–57
- managed device, 761
- managed objects, 761
- Management Information Base. *See* MIB
- managing entity, 761
- MANETs (mobile ad hoc networks), 518
- manifest file, 601, 610
- MAP message, 460
- Master Key. *See* MK
- Master Secret. *See* MS
- maximum segment size. *See* MSS
- maximum transmission unit. *See* MTU
- MBone multicast network, 411
- MCR (minimum cell transmission rate), 313
- MD5, 710
 - authentication, 389
- MDCs (modular data centers), 494
- MD5 hash algorithm, 690
- message authentication code. *See* MAC
- message digests, 707
- message integrity, 688–693, 706
 - cryptographic techniques, 675
 - digital signatures, 695
 - secure e-mail system, 707–708
- message queue, 121
- messages, 51
 - application-layer, 51
 - authenticating, 689
 - breaking into shorter segments, 51
- confidentiality, 672–673
- eavesdropping, 673
- encrypted, 672
- HTTP format, 103–108
- integrity, 673
- segmentation, 77
- semantics of fields, 97
- syntax, 96
- transmitting and receiving, 7–8
- Metcalf, Bob, 454, 470, 473
- metering function, 650
- method field, 104
- MIB (Management Information Base), 761–762, 765, 770
- MIB modules, 765, 770–772
- MIB objects, 765
- Microsoft, 64–66, 66, 114
- middleboxes, 303
- MIMO (multiple-input, multiple output) antennas, 553
- minimum cell transmission rate. *See* MCR
- minimum spanning tree, 403
- Minitel project, 63–64
- MK (Master Key), 730
- mobile ad hoc networks. *See* MANETs
- mobile devices, 81
 - Internet, 550–552
 - power management, 543–544
- mobile IP, 349, 563–569, 576, 568–569
- mobile IP standard, 562
- mobile nodes, 554–564
 - COA (care-of-address), 559
 - direct routing, 563–564
 - foreign address, 559
 - foreign network, 559
 - home network, 559
 - indirect routing, 559–562
 - location protocol, 563
 - permanent address, 559
 - permanent home, 557
 - registering with foreign agent, 566–567
 - routing to, 559–564
 - sending datagrams to correspondent, 561
 - mobile-node-to-foreign-agent protocol, 562

mobile phones and wireless LAN base stations, 548

mobile station roaming number. *See* MSRN

mobile switching center. *See* MSC

mobility, 513–514

- addressing, 556–559
- cellular network management, 570–577
- GSM (Global System for Mobile Communications), 576
- in IP subnet, 541–542
- management, 555–564
- mobile IP, 576
- network-layer functionality required, 561–562
- routing to mobile node, 559–564
- scalability, 558
- wireless networks, 575–577

mobility-related services, 513

modems, 15

modular arithmetic, 685

modular data centers. *See* MDCs

modulo arithmetic, 687–688

MPEG, 624

MPLS (Multiprotocol Label Switching), 487–490

MPLS paths, 303

MP3 (MPEG 1 layer 3), 590

MSC (mobile switching center), 550

MSDP (Multicast Source Discovery Protocol), 412

MS (Master Secret), 714, 716

MSRN (mobile station roaming number), 571–572

MSS (maximum segment size), 232–234

MTU (maximum transmission unit), 232–233

- Ethernet, 471
- IP datagrams, 335

multicast routing, 405–412

multicast addresses, 356

multicast group, 406

multicast packets, 405–412

multicast protocols, 362

multicast routing, 399, 407–412

Multicast Source Discovery Protocol. *See* MSDP

multicast trees and RTP packets, 624–625

multi-homed stub network, 397

multi-homing, 33

multi-hop wireless networks, 518

multihop paths, 263–265

multimedia

- network supported for, 632–655
- streaming stored video, 593–612
- system-level approach for delivering, 633
- VoIP (Voice-over-IP), 612–623

multimedia applications, 588

- audio properties, 590–591
- bandwidth sensitive, 92
- conversational voice, 587, 592–593
- improving quality, 635
- Internet telephony, 592–593
- streaming live audio and video, 587, 593
- streaming stored audio and video, 587, 591–592

TCP (Transmission Control Protocol), 200

- types, 591–593

UDP (User Datagram Protocol), 200–201, 282

- video over IP, 592–593
- video properties, 588–589

multipath propagation, 519

multi-player online games, 83

multiple access links, 445–461

multiple access problem, 445

multiple access protocols, 445–461

- ALOHA protocol, 63
- CDMA (code division multiple access), 449

channel partitioning protocols, 447–448

characteristics, 447–448

FDM (frequency-division multiplexing), 448

random access protocols, 447

taking-turns protocols, 447, 459–460

TDM (time-division multiplexing), 448

- multiple classes of service, 636–640
- multiple-input, multiple output antennas.
 - See* MIMO antennas
- multiplexing, 191–198
 - circuit-switched networks, 28–30
 - connectionless, 193–194
 - connection-oriented, 194–197
- multiplexing/demultiplexing service and transport layer, 198–199
- Multiprotocol Label Switching. *See* MPLS
- multi-tier hierarchy, 33
- MX record, 140, 142

- N**
- NAKs (negative acknowledgments), 207, 208
- named data, 585
- name translation and SIP (Session Initiation Protocol), 630–632
- Napster, 65
- NAT (network address translation), 306, 322–324, 349–352
 - Skype, 622
- NCP (network-control protocol), 62
- negative acknowledgements. *See* NAKs
- neighbors, 364
- neighbor-to-neighbor communication, 372–375
- Netflix, 65, 83, 588, 608–610
- Netscape Communications Corporation, 64
- network adapter, 437–438
- network address translation. *See* NAT
- network applications, 83
 - application-layer protocols, 97
 - architectures, 86–88
 - client program, 156
 - communication between client and server, 156
 - creation, 156–168
 - principles of, 84–98
 - processes communicating, 88
 - proprietary, 156–157
 - server program, 156
 - Web, 98–116

- network-assisted congestion control, 266–269
- network attacks, 55–60
- network cache servers, 106
- network-control protocol. *See* NCP
- network core
 - circuit switching, 27–32
 - network of networks, 32–35
 - packet switching, 22–27
- network dimensioning, 634–635
- network interface card. *See* NIC
- network-layer addresses, 462, 465
- network-layer components, 266
- network-layer datagram, 55
- network-layer multicast routing algorithms, 408
- network-layer packets, 51–52
- network-layer protocols
 - constrained by service model, 189
 - difficulty changing, 362
 - hosts supporting multiple, 471–472
 - IP (Internet Protocol), 190
 - logical communication between hosts, 186, 188–189
- network layer, 50–53, 189–190
 - best-effort service, 311–312
 - broadcast protocols, 405
 - complexity, 305
 - confidentiality, 718
 - connectionless service, 313
 - connection service, 313
 - connection setup, 310
 - datagram networks, 313
 - datagram service, 364
 - encapsulating transport-layer segment into IP datagram, 199
 - extracting transport-layer segment from datagram, 186
 - forwarding, 305, 308–310, 315
 - guaranteed delivery, 311
 - host-to-host communication, 305
 - host-to-host services, 313
 - ICMP (Internet Control Message Protocol), 353
 - in-order packet delivery, 311

- Internet, 332
- Internet routing protocols, 353
- IP protocol, 51–52, 332, 353
- mobility, 561–562
- passing segments to, 191–198
- path between sender and receiver, 315
- process-to-process delivery service, 191–198
- reporting errors in datagrams, 332
- reserving resources, 316
- routing, 51–52, 305–306, 308–310
- security, 705, 718–725
- services offered by, 310–313
- transport layer relationship, 186–189
- VC (virtual-circuit) networks, 313–315
- network-layer service models, 319
- network management
 - accounting management, 759
 - Comcast case study, 763–764
 - configuration management, 759
 - definition, 759
 - fault management, 758–759
 - host monitoring, 756
 - infrastructure, 760–762
 - intrusion detection, 758
 - managed device, 761
 - managed objects, 761
 - managing entity, 761
 - MIB (Management Information Base), 761–762
 - monitoring traffic, 756–758
 - network management protocol, 762
 - performance management, 758
 - security management, 759
 - SLAs (Service Level Agreements), 758
 - standards, 762
- network management agent, 762
- network management protocol, 762
- network mapping, 740
- network of networks, 32–35, 62
- network prefix, 342
- network protocols, 8–9
- networks, 340
 - access networks, 12–18
 - attacks, 57–58
- circuit-switched, 27–30
- cellular, 546–554
- components, 9–21
- crossbar, 326
- datagram, 317–319
- differentiated service, 634, 648–652
- dimensioning best-effort, 634–636
- DMZ (demilitarized zone), 741
- foreign, 557
- growth of, 62
- interconnecting, 62
- linking universities, 63
- as link layer, 486–490
- links, 635
- mobility, 513–514
- MPLS (Multiprotocol Label Switching)
 - networks, 487–490
 - multiple classes of service, 636–640
 - packet delay and loss, 35–44, 635
 - packet-switched, 4, 22–27
 - per-connection QoS (Quality-of-Service) guarantees, 652–655
 - physical media, 18–22
 - private, 718
 - programs communicating on, 84
 - routes, 4
 - scheduling mechanisms, 640–645
 - security, 55–56, 671–674
 - sockets, 89–90
 - switched local area networks, 461–486
 - VC (virtual-circuit), 314–317
 - visited, 557
 - wireless LANs, 515, 518, 526–546
- network service models, 310–313
- NEXT-HOP attribute, 394–395, 397
- NIC (network Interface card), link layer implementation, 437–438
- NI (no increase) bit, 268–269
- nmap port scanning tool, 196, 258
- NOC (network operations center), 755
- nodal processing, 36
- no increase bit. *See* NI bit
- nomadic computing, 81

nonces, 704–705, 716–717
 non-persistent connections, 100–103, 198
 nonpreemptive priority queuing discipline, 643
 non-real-time applications, 92–93
 nonrepudiation and cryptographic techniques, 675
 nslookup program, 141–142
 N-way-unicast, 400–401

O

OBJECT IDENTIFIER data type, 766
 objects, 99, 103
 OC (Optical Carrier) standard link, 21
 odd parity schemes, 440
 offered load, 261
 OLT (optical line terminator), 16
 ONT (optical network terminator), 16
 Open-Shortest Path First. *See OSPF*
 Open Systems Interconnection model. *See OSI model*
 operational security, 673, 731–742
 Optical Carrier standard link. *See OC standard link*
 optical distribution network, 15–16
 optical line terminator. *See OLT*
 optical network terminator. *See ONT*
 options field, 235
 origin authentication, 363
 orthogonal frequency division multiplexing. *See OFDM*
 OSI (Open Systems Interconnection) model, 52–53
 OSPF (Open-Shortest Path First), 366–367, 384, 388–390, 498
 LSAs (link-state advertisements), 405
 router authentication, 388–389
 OS vulnerability attacks, 740
 out-of-band, 117
 out-of-order segments, 236
 output buffers, 25
 output ports, 320–326
 packet queues, 327–331
 packet scheduler, 329
 processing, 326

output queue, 25
 overlapping fragments, 338
 overlay network, 154, 486

P

packet classification, 648–649
 packet delay, 35–44, 102, 635
 packet-discard policy, 641
 packet filtering, 732, 737
 packet forwarding, 26–27
 packet loss, 25, 41–42, 91, 259, 613, 635
 error concealment, 620–621
 FEC (forward error correction), 618–619
 interleaving, 618–620
 predicting imminent, 278
 recovering from, 213, 618–621
 packet marking, 638
 packet-radio networks, 62, 511
 packet repetition, 621
 packets, 4, 22
 average rate, 645
 bit errors, 207
 buffering, 218
 burst size, 646
 controlled flooding, 401–403
 cumulative acknowledgment, 222
 delays, 36–37
 delivering, 204–205
 destination, 35
 destination IP address, 158
 detecting loss, 212–215
 dropping, 41, 329
 duplicate, 210, 213–214
 duplicate ACKs, 211
 end-to-end delay, 612
 FIFO (first-in-first-out), 641–642
 format of, 5
 forwarding, 4, 308–310, 321–322, 649
 header fields, 55
 IP spoofing, 59–60
 jitter, 614
 moving between nodes, 52
 path, 35
 payload fields, 55

- peak rate, 646
- prefix destination address, 318
- priority queuing, 642–643
- processing delays, 36–37
- queuing delays, 25, 37, 39–42
- round-robin queuing, 643–644
- round-trip delays, 42–43
- routing, 308–310
- RTT (round-trip time), 102–103
- same priority class, 642
- sender's source address, 158
- sending, 24
- sending multiple, 218
- sequence numbers, 210, 218, 220
- sockets, 158
- source, 35
- source address, 162
- switching, 324–326
- tracing, 42–43
- transmitting, 213–214
- uncontrolled flooding, 401
- VC number, 315
- WFQ (weighted fair queuing), 644–645
- what to do when loss occurs, 212–215
- where queuing occurs, 327–331
- Packet Satellite, 511
- packet scheduler, 329
- packet sniffers, 58–59, 78
- packet-switched networks, 4, 25
 - ARPAnet, 61
 - comparing transmission and propagation delay, 38–39
 - delays, 35–39
 - end-to-end delays, 42–44
 - packet loss, 41–42
 - processing delays, 36–37
 - propagation delays, 37–38
 - queuing delays, 37, 39–42
 - sending packets, 28
 - transmission delays, 37
- packet switches, 4, 310
 - facilitating exchange of data, 6
 - link-layer switches, 22, 53, 310
 - output buffers, 25
- routers, 22, 53, 310
- store-and-forward transmission, 22, 24
- packet switching, 30–31
 - alternative to circuit switching, 60
 - forwarding tables, 26–27
 - packet loss, 25
 - queuing delays, 25
 - queuing theory, 60
 - routing protocols, 26–27
 - secure voice over military networks, 60
 - store-and-forward transmission, 22, 24
 - VC (virtual-circuit) approach, 267
- packet-switching networks, 62
- paging, 550
- parity checks, 440–442
- passive optical networks. *See* PONs
- passive spanning, 530
- passwords, 703, 710
- paths, 4, 365
 - least-cost, 365
 - multihop, 263–265
- payload fields, 55
- PBXs (private branch exchanges), 627
- PCM μ -law, 628–629
- PCM (pulse code modulation), 590
- PDU and SNMP applications, 776–777
- peak rate, 646
- peer, 34
- peer churn and DHTs (distributed hash tables), 155–156
- peering, 33
- peers, 86, 144–145
 - DHT, 155–156
 - file sharing, 145–151
 - torrent, 149
- peer-to-peer applications. *See* P2P applications
- per-connection QoS (Quality-of-Service) guarantees, 634, 652–655
- per-connection throughput, 260
- perfectly reliable channel, 206–207
- Performance Management, 758, 763–764
- per-hop behavior. *See* PHB

- permanent address, 559
- persistent connections, 100–103
- persistent HTTP, 198
- personal area networks
 - Bluetooth, 544–545
 - Zigbee, 545–546
- PGP (Pretty Good Privacy), 678, 706, 709–711
- PHB (Per-hop behavior), 649–651
- physical address, 463
- physical layer, 50, 52–53
- physical media, 4
 - coaxial cable, 20
 - costs, 19
 - fiber optics, 20–21
 - guided media, 19
 - radio channels, 21
 - satellite radio channels, 21–22
 - twisted-pair copper wire, 19–20
 - unguided media, 19
- piconet, 544
- piggybacked acknowledgment, 237
- PIM (Protocol-Independent Multicast), 411–412, 584
- ping program, 353
- pipelined reliable data transfer protocols, 215–218
- pipelining, 218
 - persistent connections, 103
 - TCP (Transmission Control Protocol), 240
- plaintext, 675
- playback attacks, 703, 777
- playout delay, 614
- plug-and-play protocol, 346
- plug-and-play switches, 479–480
- PMS (Pre-Master Secret), 716
- points of presence. *See* PoPs
- point-to-point, 232
- point-to-point communication link, 434
- point-to-point links, 436, 445
- Point-to-Point Protocol. *See* PPP
- poisoned reverse, 377–378
- poisoning attack, 143
- policing disciplines, 645–648
- policing mechanisms, 640
- polling protocols, 459
- polls, 459
- polyalphabetic encryption, 678
- polynomial codes, 443
- PONs (passive optical networks), 15–16
- POP3 (Post Office Protocol-Version 3), 127–129
- POP3 server, 127, 129
- PoPs (points of presence), 33
- POP3 user agent, 128
- port-based VLAN, 483–484
- port numbers, 90, 158
 - addressing processes, 351
 - destination, 234
 - protocols, 90
 - source, 234
 - Web servers, 197–198
 - well-known, 192, 196
- port-based VLAN, 483–484
- port scanners, 196
- port scans, 196, 740
- positive acknowledgment. *See* ACK
- POST method, 104–105
- Post Office Protocol-Version 3. *See* POP3
- power management, 543–544
- P2P (peer to peer)
 - applications, 97–98
 - architecture, 86–88, 144–148
 - BitTorrent, 149–151
 - connection reversal, 352
 - DHTs (distributed hash tables), 145, 151–156
 - file distribution, 83, 88, 45–151
 - NAT, 351–352
 - Skype, 621–622
 - video streaming applications, 592
- PPP (point-to-point protocol), 434, 445
- PPstream, 87
- PPTV and P2P delivery, 611
- prefetching, 592
 - video, 596–597
- prefixes, 342, 344, 393
 - awareness of, 396–397
- BGP attributes, 394

- forwarding table, 396–397
- gateway routers, 393
- Pre-Master Secret. *See* PMS
- prerecorded video, 591
- presentation layer, 53
- presentation service, 780
- Pretty Good Privacy. *See* PGP
- priority queuing, 642–643
- privacy
 - cookies, 108
 - proxy servers, 738
 - QQ, 623
 - Skype, 623
 - SSL (Secure Sockets Layer), 738
 - Web sites, 738
- private branch exchanges. *See* PBXs
- private CDNs (Content Distribution Networks), 603
- private key, 684–685, 693, 708
 - passwords, 710
- private networks, 66, 718
- processes, 88–90
 - communicating by sending messages to sockets, 157
 - communicating using UDP sockets, 158
 - connection sockets, 198
 - handshaking, 231
 - logical communication between, 186
 - sockets, 191
- processing delays, 36–37
- programming, event-based, 223
- propagation delays, 24, 36–39, 456
- proprietary network applications, 156–157
- Protocol-Independent Multicast. *See* PIM
- protocols, 5, 68
 - alternating-bit, 214
 - application-layer, 49–50
 - congestion-control, 9
 - defining, 7–9
 - hardware-implemented, 9
 - human analogy, 7–8
 - interior gateway, 384
 - Internet, 9
- IP (Internet Protocol), 5
- layering, 49–50
- nonce, 704–705
- packet sizes, 335
- plug-and-play, 346
- port numbers, 90
- real-time interactive applications, 623–632
- routing, 51–52
- RTP, 623–626
- SIP, 626–632
- soft state, 408–409
- SR (selective repeat), 223–230
- stateless, 100
- stop-and-wait, 209, 215, 217
- TCP, 5
- UDP, 5
- transmission and receipt of messages, 7–8
- transport-layer, 50
- protocol stack, 50
- provider, 32
- proxy servers, 106, 110, 738
- public key algorithm, 716
- public key certification, 697–699
- public-key cryptography, 708
 - digital signatures, 693–694
 - private key, 693
 - public key, 693, 697
 - secure e-mail system, 706–707
- public key encryption, 683–688
- public-key encryption algorithm, 687
- public keys, 684–685, 693, 706–708, 713–714
 - binding to particular entity, 697–698
 - certifying, 708
 - encryption/decryption algorithms, 684
- public key systems, 676
- pull protocol, 124
- pulse code modulation. *See* PCM (pulse code modulation)
- push protocol, 124
- PUT method, 105
- Python, 157, 160, 193

Q

QAM16 modulation, 521
 Q2931b protocol, 654
 QoS (Quality-of-Service), 329, 653–654
 QQ, 592, 623
 Quality-of-Service. *See* QoS
 quantization, 590
 query
 ARP message, 467
 information about, 141
 query messages, 140–142
 queues
 FIFO (first-in-first-out), 641–642
 packet-dropping policy, 641
 priority queuing, 642–643
 provable maximum delay, 647–648
 round robin queuing discipline,
 643–644
 WFQ (weighted fair queuing),
 644–645
 work-conserving round robin
 discipline, 644
 queuing, 327–331
 queuing delays, 25, 36–37, 39–42, 60

R

radio channels, 21
 Radio Network Controller. *See* RNC
 RADIUS protocol, 530, 730
 random access protocols, 447, 473
 Aloha protocol, 452–453
 CSMA (carrier sense multiple access)
 protocol, 453–456
 CSMA/CD (carrier sense multiple
 access with collision detection),
 455–459
 slotted ALOHA protocol, 450–452
 Random Early Detection algorithm. *See*
 RED algorithm
 rarest first, 149
 rate adaptation, 542–543
 RC4 algorithm, 727–728
 RCP (Routing Control Platform), 786
 rdt (reliable data transfer protocol), 204
 building, 206–215

packet reordering, 229–230
 pipelined, 215–218
 TCP (Transmission Control Protocol),
 204
 unreliable layer below, 204
 real-time applications
 timing, 92
 UDP (User Datagram Protocol), 200
 real-time interactive applications
 protocols, 623–632
 RTP (Real-Time Transport Protocol),
 623–626
 SIP, 626–632
 real-time measurements of delay and loss
 performance, 606
 Real-Time Streaming Protocol. *See* RTSP
 Real-Time Transport Protocol. *See* RTP
 receive buffer, 233
 receiver authentication, 706
 receiver-based recovery, 621
 receiver feedback, 208
 receivers
 ACK generation policy, 247
 defining operation, 206
 sequence number of packet acknowl-
 edged by ACK message, 212
 receiver-side transport layer, 54
 receive window, 250–252
 receive window field, 234
 receiving adapter, 472
 receiving processes addresses, 90
 records, inserting in DNS database, 142,
 144
 recursive queries and DNS servers,
 137–138, 140
 RED (Random Early Detection)
 algorithm, 329
 regional ISPs, 33
 registrars, 142
 registration with home agent, 568–569
 relays, 622–623
 reliable channel, 204
 reliable data transfer, 91, 190
 application layer, 204
 channel with bit errors, 207–212

- link layer, 204
- lossy channel with bit errors, 212–215
- perfectly reliable channel, 206–207
- principles, 204–230
- reliable channel, 204
- TCP (Transmission Control Protocol), 230–231, 240, 242–250
- transport layer, 204
 - transport-layer protocols, 91
- reliable data transfer protocol. *See* rdt
- reliable data transfer service, 235
- reliable delivery, 436
- reliable transport service, 269
- remote host, transferring files, 116–118
- rendezvous point, 404
- repeater, 474
- replicated servers, 132
- reply messages and DNS (domain name system), 140–142
- repositioning video, 600
- request messages and HTTP, 103–105
- request-response mode, 772
- requests for comments. *See* RFCs
- Request to Send control frame. *See* RTS control frame
- residential ISPs, 87
- resource-management cells. *See* RM cells
- resource records. *See* RRs
- resource reservation protocols, 362
- resources
 - admitting or blocking flows, 653
 - efficient use of, 640
 - reservations, 653–654
- response ARP, 467
- response messages and HTTP, 105–108
- retransmission, 208, 212
- retransmitting data, 241, 262
- retransmitting packets, 259, 261–263
- reverse path broadcast. *See* RPB
- reverse path forwarding. *See* RPF
- Rexford, Jennifer, 786–787
- RFCs (requests for comments), 5
- RIP advertisements, 384–385
- RIP request message, 387
- RIP response message, 384
- RIP routers, 386–387
- RIP (Routing Information Protocol), 384, 498
 - hops, 384
 - implementation aspects, 386–388
 - IP network-layer protocol, 387
 - lower-tier ISPs, 388
 - modifying local routing table and propagating information, 387
 - RIP messages, 384–385
 - RIP table, 385–386
 - routing updates, 384
 - UDP transport-layer protocol, 387
 - UNIX implementation, 387–388
- Rivest, Ron, 684, 690
- RM (resource-management cells), 267–269
- RNC (Radio Network Controller), 552
- roaming number, 572
- Roberts, Larry, 61, 511
- root DNS servers, 134–136
- round robin queuing discipline, 643–644
- round-trip delays, 43
- round-trip time. *See* RTT
- route aggregation, 342
- route attributes, 395
- router control plane functions, 322
- router discovery message, 566–567
- router forwarding plane, 321
- routers, 4, 12, 22, 53, 303, 310
 - access control lists, 734
 - adapters, 468
 - address of, 43
 - administrative autonomy, 380
 - area border, 389
 - ARP modules, 468
 - AS-PATH attribute, 394
 - ASs (autonomous systems), 380
 - authenticated and encrypted channel between, 725
 - buffering packet bits, 24
 - buffer sizing, 328–329
 - connected into network, 338
 - connection state information, 315
 - control functions, 321–322

- routers (*continued*)
- control plane implemented in, 331
 - data center hierarchy, 492–493
 - default, 364
 - destination, 364
 - finite buffers, 261–265
 - firewalls, 355, 481
 - first-hop, 364
 - fixed-length labels, 487
 - forwarding function, 320–322
 - forwarding table, 26, 308–309, 317–318, 322–323, 394, 396–397, 469
 - gateway, 380–381
 - implementing layers 1 through 3, 53
 - incident links, 22
 - input ports, 320
 - input processing, 322–324
 - interfaces, 338, 468
 - intra-AS routing protocols, 397
 - IP addresses, 394, 465, 468
 - IP protocol, 53
 - label-switched, 488
 - layer-2 packet switch, 480
 - link-layer and MAC addresses, 462–463, 465
 - longest prefix matching rule, 318–319
 - lookup, 323–324
 - looping advertisements, 394
 - memory access times, 324
 - network core, 4
 - network-layer addresses, 462, 465
 - output ports, 320–321
 - output processing, 326
 - packet-forwarding decisions, 364
 - packet loss, 327
 - packets not cycling through, 481
 - physical links between, 364
 - plug-and-play, 481
 - primary role, 306
 - processing datagrams, 480
 - processing packets, 351
 - protocols, 9
 - queuing, 327–331
 - routing control plane, 331
 - routing packets, 380–382
 - routing processor, 321
 - routing tables, 385–386
 - scale, 379–380
 - self-synchronizing, 371
 - source, 364
 - spanning tree, 481
 - store-and-forward, 22, 24
 - store-and-forward packet switches, 480
 - versus* switches, 480–482
 - switching, 320, 324–326
 - terminating incoming physical link, 320
 - VC setup, 316
 - routes, 4, 394–396
 - route summarization, 342
 - routing, 305–306, 308–310
 - advertising information, 382–383
 - broadcast, 399–405
 - calls to mobile user, 571–572
 - distance vector, 384
 - hierarchical, 379–383
 - hot-potato, 382
 - to mobile node, 559–564
 - multicast, 399, 405–412
 - storing information, 379–380
 - routing algorithms, 309, 363–383
 - ARPAnet, 366
 - circuit-switched, 379
 - decentralized, 366
 - DV (distance-vector) algorithm, 366, 371–379
 - dynamic, 366
 - forwarding tables, 364
 - global, 365–366
 - hierarchical routing, 379–383
 - least costly paths, 365
 - load-sensitive, 366
 - LS (link-state) algorithms, 366–371
 - path from source to destination router, 364
 - scale of routers, 379–380
 - static, 366
 - switches, 494–495
 - viewing packet traffic flows, 379
 - routing control plane, 331

Routing Control Platform. *See* RCP
 routing daemons, 674
 Routing Information Protocol. *See* RIP
 routing loop, 377
 routing protocols, 26–27, 51–52
 BGP (Border Gateway Protocol), 390–399, 498–499
 DV (distance vector) algorithms, 374–375
 executing, 321
 inter-AS, 382
 Internet, 383–399
 intra-AS, 380–381
 IS-IS, 384
 messages, 309
 OSPF (Open-Shortest Path First), 384
 RIP (Routing Information Protocol), 384
 RPB (reverse path broadcast), 402
 RPF (reverse path forwarding), 402–403, 411
 RRs (resource records), 139–141
 RSA algorithm, 684–688, 710
 RST flag bit and segment 235, 258
 RSVP, RSVP-TE protocol, 489, 654
 RTP packets, 624–625
 RTP (Real-Time Transport Protocol), 588, 623–626, 668
 UDP streaming, 595
 RTS/CTS exchange, 537
 RTS frame, 536–537
 RTSP (Real-Time Streaming Protocol), 117, 595, 668
 RTS (Request to Send) control frame, 535–537
 RTT (round-trip time), 102–103
 EWMA (exponential weighted moving average), 240
 TCP (Transmission Control Protocol), 238–241

S

SAD (Security Association Database), 721
 SA (security association), 720–721
 satellite links, 16, 21–22

scalability and P2P architecture, 145–148
 scheduling mechanisms, 640–645
 Schulzrinne, Henning, 623, 632, 668–670
 SDN (Software Defined Networking), 786
 secure communication, 672–674
 secure e-mail system, 706–708
 Secure Hash Algorithm. *See* SHA-1
 secure networking protocols and message integrity, 689
 Secure Network Programming, 511
 Secure Sockets Layer. *See* SSL
 security, 55–56
 application-layer protocol, 705
 attacks, 674
 cryptography, 675–688
 data link layer, 705
 digital signatures, 688–699
 e-mail, 705–711
 end-point authentication, 700–705
 IEEE 802.11i, 728–731
 IP datagrams, 718
 IP (Internet Protocol), 362–363
 IPsec, 362
 message integrity, 688–693
 mobile IP, 566
 network layer, 705, 718–725
 networks, 671–674
 operational, 673, 731–742
 OSPF (Open-Shortest Path First), 388–389
 P2P architecture, 88
 public key encryption, 683–688
 RSA, 687
 SNMPv3, 775–778
 switches, 479
 TCP connection, 711–717
 transport-layer protocols, 93, 705
 transport services, 93
 user-based, 777
 WEP (Wired Equivalent Privacy), 726–728
 wireless LANs, 726–731
 security and administration capabilities, 765
 security association. *See* SA

- Security Association Database. *See* SAD
 Security Management, 759, 764
 Security Policy Database. *See* SPD
 segments, 51, 186, 189
 - acknowledgment number, 236
 - destination port number field, 192
 - fast retransmit, 248
 - fields, 191–192
 - out-of-order, 236
 - piggybacked acknowledgment, 237
 - sequence numbers, 235–238
 - source port number field, 192
 - TCP (Transmission Control Protocol), 233
 - unique identifiers, 192
 selective acknowledgment, 250
 selective repeat protocols. *See* SR protocols
 self-learning, 478–479, 497, 542
 self-replicating, 56
 self-scalability, 87
 send buffer, 232
 sender
 - countdown timer, 214
 - defining operation, 206
 - detecting and recovering from lost packets, 212–215
 - leftmost state, 208
 - receive window, 250
 - rightmost state, 208
 - sending multiple packets without acknowledgments, 218
 - sequence number of packet, 212
 - utilization, 217
 sender authentication, 706–708
 sender-to-receiver channel, 213–214
 sending rates, 260
 send side states rdt2.0 protocols, 208
 sequence-number-controlled flooding, 401–403, 405
 sequence numbers, 210, 212, 218–220, 234, 614–615, 618, 717
 IPsec, 724
 RTP packets, 625
 SSL (Secure Sockets Layer), 715
 SYN segment, 252–253
 TCP segments, 235–236
 TCP (Transmission Control Protocol), 244, 249
 Telnet, 237–238
 server authentication, 712
 server processes, 88, 164, 232
 server program, 156, 163
 servers, 2, 10–11, 88–89
 - always on, 86
 - dedicated socket, 167
 - hostname of, 160
 - IP addresses, 86, 160, 161, 163
 - network attacks, 57–58
 - non-persistent connections, 198
 - persistent HTTP, 198
 - port number, 161, 167
 - TCP socket creation, 167
 - Web caches as, 111
 server SMTP, 122
 server socket TCP connection, 163
 server-to-client throughput, 44–45
 Service Level Agreements. *See* SLAs
 service model, 49
 service providers and private networks, 66
 services, 49
 - description of Internet, 5–7
 - DNS (domain name system), 131–133
 - flow of packets, 311
 - transport layer, 186
 - transport protocols, 189
 Service Set Identifier. *See* SSID
 Serving GPRS Support Nodes. *See* SGSNs
 session encryption key, 714
 Session Initiation Protocol. *See* SIP
 session keys, 687, 707, 714
 session layer, 53
 SGMP (Simple Gateway Monitoring Protocol), 764
 SGSNs (Serving GPRS Support Nodes), 552
 SHA, 710
 Shamir, Adi, 684

Shannon, Claude, 80, 82
 shared medium, 20
 SHA-1 (Secure Hash Algorithm), 691
 shortest paths, 365
 SIFS (Shorter Inter-frame Spacing), 532
 signaling messages, 316
 signaling protocols, 317
 signal-to-noise ratio. *See SNR*
 signature-based IDSs (intrusion detection systems), 741–742
 silent periods, 29–30
 simple authentication, 389
 Simple Gateway Monitoring Protocol. *See SGMP*
 Simple Mail Transfer Protocol. *See SMTP*
 Simple Network Management Protocol. *See SNMP*
 single-hop, wireless networks, 518
 SIP (Session Initiation Protocol), 588, 626–632, 668–669
 Skype, 65, 83, 87, 588, 621–623
 conversational voice and voice, 592
 proprietary application-layer protocols, 97
 UDP (User Datagram Protocol), 613
 SLAs (Service Level Agreements), 758
 sliding-window protocol, 220
 slotted ALOHA protocol, 450–452
 node’s decision to transmit, 453–455
 small office, home office subnets. *See SOHO subnets*
 SMI (Structure of Management Information), 765, 766–769
 SMTP clients, 122–123
 SMTP servers, 123
 SMTP (Simple Mail Transfer Protocol), 51, 97, 117, 120–127
 SNMP applications, 776–777
 SNMP messages, 777
 SNMP (Simple Network Management Protocol), 758–759, 762, 764–778
 SNMPv3, 765, 775–778
 SNMPv2 (Simple Network Management Protocol version 2), 772, 773–775

Snort IDS system, 740–742
 SNR (signal-to-noise ratio), 520–521
 social networking, 83, 86
 social networks, 64–65, 100
 socket interface, 100
 socket module, 160
 socket programming
 TCP (Transmission Control Protocol), 158, 163
 UDP, 157–158
 sockets, 89–91, 91, 191
 assigning port number, 162
 port number, 158
 soft guarantee, 634
 soft state protocols, 408–409
 software control plane, 331
 Software Defined Networking. *See SDN*
 SOHO (small office, home office) subnets
 and IP addresses, 349–352
 source
 host and source router, 364
 total delay to destination, 42–44
 source port numbers, 192, 194, 196, 234
 source quench message, 353
 source router, 364
 source-specific congestion-control actions, 267
 source-specific multicast. *See SSM*
 spam, 56
 spanning-tree broadcast, 403–405
 spanning trees, 403–405, 481
 spatial redundancy, 589
 SPD (Security Policy Database), 724
 special socket server program, 163
 speed-matching service, 250
 SPI (Security Parameter Index), 721
 split-connection approaches, 577
 Sprint, 5, 33, 758
 spyware, 56
 SRAM, 324
 SR (selective repeat) protocols, 223–230
 SSH protocol, 237
 SSID (Service Set Identifier), 529
 SSL record, 715–716

- SSL (Secure Sockets Layer), 711
 - anonymity, 738
 - API (Application Programmer Interface) with sockets, 712
 - block ciphers, 678
 - breaking data stream into records, 714
 - connection closure, 717
 - cryptographic algorithms, 716
 - data transfer, 713–715
 - designed by Netscape, 711
 - handshake, 713–714, 716–717
 - HTTP transactions security, 712
 - key derivation, 713–714
 - nonces, 717
 - popularity, 711
 - privacy, 738
 - public key certification, 697
 - sequence numbers, 715
 - SSL classes/libraries, 712
 - SSL record, 715–716
 - transport protocols, 712
- SSM (source-specific multicast), 412
- state, 117
- stateful packet filters, 732, 735–736
- stateless protocols, 100
- static routing algorithm, 366
- stations, 531–533
- status line in HTTP response messages, 106
- steaming prerecorded videos, 591
- stop-and-wait protocols, 209–210, 215, 217
- store-and-forward packet switches, 22, 24, 480
- stream ciphers, 678
- streaming, 591
 - live audio and video, 587, 593
 - stored audio and video, 587, 591–592
 - video, 589
- streaming stored video, 593–612
 - adaptive HTTP streaming, 593
 - adaptive streaming, 600–601
 - bandwidth, 594
- CDNs (content distribution networks), 602–608
- client buffering, 594–595
- continuous playout, 591–592
- DASH (Dynamic Adaptive Streaming over HTTP), 600–601
- end-to-end delays, 594
- HTTP streaming, 593, 596–600
- interactivity, 591
- KanKan, 611–612
- Netflix, 608–610
- streaming, 591
- UDP streaming, 593, 595–596
- YouTube, 610–611
- streaming video, 592
 - TCP (Transmission Control Protocol), 596
- Structure of Management Information. *See* SMI
- stub network, 397–398
 - multi-homed, 397
- subnet mask, 340
- subnets, 340
 - advertising existence to Internet, 391
 - class A, B and C networks, 344
 - defining, 341
 - DHCP offer message, 347
 - DHCP servers, 346
 - IP addresses, 340, 342, 345
 - IP definition of, 340–341
 - prefixes, 393
 - sending datagrams off, 468–469
 - shortest-path tree, 388
- successful slots, 451
- switched Ethernet, 470
- switched-LANs
 - ARP (Address Resolution Protocol), 465–468
 - Ethernet, 469–476
 - link-layer addressing, 462–469
 - link-layer switches, 476–482
 - MAC addresses, 463–465
 - switch poisoning, 480
 - VLANs (virtual local area networks), 482–486
- switched networks, 481

- switches, 80
 aging time, 478
 broadcasting frames, 464
 broadcast storms, 481
 collisions elimination, 479
 congestion-related information, 268
 data center hierarchy, 492–493
 enhanced security, 479
 Ethernet, 470, 475
 filtering, 476–477
 filtering frame, 477
 forwarding, 476–477
 gathering statistics, 479
 heterogeneous links, 479
 high filtering and forwarding rates, 480
 link-layer, 461, 476–482
 link-layer addresses, 462
 link-layer frames, 476
 MAC addresses, 480
 management, 479
 plug-and-play devices, 479–480
 processing frames, 480
versus routers, 480–482
 routing algorithms, 494–495
 self-learning, 478–479, 497, 542
 small networks, 482
 store-and-forward packet switches,
 480
 switch table, 476
 tracking behavior of senders, 267
 transparent, 476
 trunk port to interconnect, 484
 VLANs (virtual local area
 networks), 483–484
 switch fabric, 320, 322, 327, 329–330
 switching and routers, 324–326
 switch output interfaces buffers, 476
 switch poisoning, 480
 switch table, 476–477
 symmetric algorithm, 716
 symmetric key, 706–707, 707
 symmetric key algorithm
 block ciphers, 678–681
 Caesar cipher, 676
 monoalphabetic cipher, 676–677
 polyalphabetic encryption, 678
 stream ciphers, 678
 symmetric key encryption and CBC
 (cipher-block chaining), 681–682
 SYNACK segment, 257–258
 SYN bit, 235, 253
 SYN cookies, 257
 SYN flood attack, 252, 253, 257
 SYN packet, 258
 SYN segments, 252–254, 257–258
 SYN_SENT state, 254
- T**
- taking-turns protocols, 447, 459–460
 TCAMs (Ternary Content Address
 Memories), 324
 TCP buffers, 597–598
 TCPClient.py client program, 164–166
 TCP clients, 195, 253–255
 TCP congestion-control algorithm,
 272–277, 279
 TCP connections, 57, 94
 allocating buffers and variables, 253
 bandwidth, 281
 bottleneck link, 279–281
 buffers, 233
 between client and server, 166
 client process, 232
 client-side TCP sending TCP
 segment to server-side TCP,
 252–253
 client socket, 163
 connection-granted segment, 253
 ending, 253–254
 establishing, 232, 252–253, 713
 full-duplex service, 232
 HTTP server, 596
 management, 252–256, 258
 out-of-order segments, 236
 packet loss, 281
 parallel and fairness, 282
 point-to-point, 232
 processes sending data, 232–233
 receive buffer, 233, 250
 regulating rate of traffic, 190

- TCP connections (*continued*)
 - security, 711–717
 - send buffer, 232
 - server process, 232
 - server socket, 163
 - socket connection to process, 233
 - split-connection approaches, 577
 - three-way handshake, 102–103, 166, 232
 - throughput, 280
 - transporting request message and response message, 101
 - variables, 233
- TCP header, 234–235
- TCP/IP (Transmission Control Protocol/Internet Protocol), 5, 63, 93, 231, 431
- TCP ports, 258
- TCP Reno, 276, 278
- TCP segments, 233–236, 253
 - with different source IP addresses, 194–195
 - header overhead, 200
 - loss, 266
 - reordering, 715
 - structure, 233–238
- TCP sender, 242–243, 269, 270
 - awareness of wireless links, 577
 - congestion control, 250
- TCP server, 163, 195
- TCPServer.py server program, 166–168
- TCP sockets, 165–166, 497, 499
 - server-side connection socket, 163
 - welcoming socket, 163
- TCP splitting, 273
- TCP streaming and prefetching video, 597
- TCP SYNACK segment, 499
- TCP SYN segment, 499
- TCP Tahoe, 276
- TCP (Transmission Control Protocol), 5, 51, 93, 189, 313, 338
 - acknowledgment numbers, 244
 - block ciphers, 678
 - buffer and out-of-order segments, 249
 - buffer overflow, 251
- byte stream, 242
- checksum, 334
- client-server application, 157
- congestion avoidance, 272–276
- congestion control, 95, 190, 199–200, 240, 247, 269–272, 274–283, 576–577, 596, 613
- congestion window, 269–270, 276–277, 576
- connection-establishment delays, 200
- connection-oriented, 94, 163, 230–238
- connection state, 200, 231
- continued evolution of, 279
- cumulative acknowledgments, 236, 243, 248–249
- duplicate ACK, 247–248
- early versions, 62
- end-to-end congestion control, 266, 269
- extending IP’s delivery service, 190
- fairness, 279–282
- fast retransmit, 247–248
- flow control, 240, 250–252
- full-duplex, 235
- GBN (Go-Back-N) protocol, 248–250
- high-bandwidth paths, 279
- host-based congestion control, 63
- HTTP and, 116, 200
- implicit NAK mechanism, 240
- integrity checking, 190
- Internet checksum, 442
- lost acknowledgment, 244
- lost segments, 238
- MSS (maximum segment size), 232–234
- MTU (maximum transmission unit), 232–233
- multimedia applications, 200
- negative acknowledgments, 248
- packet loss, 247–248, 613
- pipelining, 240
- positive acknowledgments, 240
- receive buffer, 270
- receiver-so-sender ACK, 576
- receive window, 251

- reliable data transfer, 96, 190, 230–231, 240
- reliable data transfer service, 95, 100, 123, 163, 199–200, 235, 242–250
- resending segment until acknowledged, 199
- retransmission timeout interval, 241
- retransmission timer, 242
- retransmitting data, 473
- retransmitting segments, 239–240, 246, 249, 575–576
- RST segment, 258
- RTT (round-trip time) estimation, 238–241
- security services, 95
- segments, 189
- selective acknowledgment, 250
- separation of IP, 62
- sequence numbers, 244, 249
- server-to-client transmission rate, 596
- services, 94–95
- socket programming, 158, 163
- states, 254
- state variable, 243
- steady-state behavior, 278–279
- streaming media, 200–201
- streaming video, 596
- SYNACK segment, 258
- SYN segments, 257–258
- TCP Reno, 276, 278
- TCP segments, 233
- TCP Tahoe, 276
- TCP Vegas, 278
- 32-bit sequence number, 220
- three-way handshake, 163, 200, 253
- throughput macroscopic description, 278–279
- timeout, 238–241, 243
- timeout, 244–247
- timeout/retransmit mechanism, 238
- transmission rate, 278
- Web servers, 197–198
- window size, 266
- wireless networks, 575–577
- TCP Vegas, 278
- TDM (time-division multiplexing), 28–30, 31, 448, 549
- telco (telephone company), 13–14
- Telenet, 62
- telephone company. *See* telco
- telephone networks, 27
- circuit switching, 60
 - complexity, 319
 - frequency band, 29
 - packet switching, 31
- Telnet, 86
- blocked, 737
 - sending message to mail server, 125
 - SMTP server, 124
 - TCP example, 234, 237–238
- temporary IP address, 346
- 10BASE-2, 473–474
- 10BASE-T, 473–474
- 10GBASE-T, 474–475
- Ternary Content Address Memories. *See* TCAMs
- 3GPP (3rd Generation Partnership Project), 550, 552, 362
- third-party CDNs (Content Distribution Networks), 603
- 3DES, 680
- 3G cellular data networks, 550–552
- 3G cellular mobile systems *versus* wireless LANs, 548
- 3G core network, 550–552
- 3G networks, 669
- 3G radio access networks, 552
- 3G systems, 547
- 3G UMTS and DS-WCDMA (Direct Sequence Wideband CDMA), 552
- three-way handshake, 102–103, 232, 253, 499, 735
- throughput, 260
- average, 44
 - end-to-end, 44–47
 - fluctuations in, 92
 - instantaneous, 44
 - macroscopic description for TCP, 278–279
 - server-to-client, 44–45

- throughput (*continued*)
 - streaming video, 592
 - TCP connection, 280
 - transmission rates of links, 47
 - transport-layer protocols, 92
 - zero in heavy traffic, 265
- tier-1 ISPs, 33–34
- time-division multiplexing. *See* TDM
- timeout
 - doubling interval, 246–247
 - event, 222, 244
 - length of intervals, 238–239
 - setting and managing interval, 241
 - TCP (Transmission Control Protocol), 238–241, 243
- timer management and overhead, 242
- time-sensitive applications, 95
- time-sharing networks, 62
- time slots, 448
- timestamps, 614–615, 617, 625
- time-to-live field. *See* TTL (time-to-live) field
- timing guarantees, 92–93
- TLD (top-level domain) DNS servers, 134–136, 143
 - DNS servers, 134
- TLS (Transport Layer Security), 711
- TLV (Type, Length, Value) approach, 780
- token-passing protocol, 459–460
- top-down approach, 50
- top-level domain DNS servers. *See* TLD
 - DNS servers
- top-level domains, 135
- Top of Rack switch. *See* TOR switch
- top-tier switch, 492
- TOR anonymizing and privacy service, 738
- torrents, 149
- TOR (Top of Rack) switch, 490, 492
- TOS (type of service) bits, 333
- total nodal delay, 36
- Traceroute program, 27, 353–355
 - end-to-end delays, 42–43
- tracker, 149
- traditional packet filters, 732–734
- traffic
 - bursty, 60
 - conditioning, 648–649
 - intensity, 40
- traffic engineering, 489
- traffic isolation, 638–640
- traffic policing, 638–639
- traffic profile, 650
- transferring files, 116–118
- transfer time, 45
- Transmission Control Protocol. *See* TCP
- Transmission Control Protocol/Internet Protocol.
- transmission delays, 36–39
- transmission rates, 4, 45–46
- transmitting
 - frames, 532
 - packets in datagram networks, 317
- transport layer, 51, 53, 185
 - application-layer message, 54
 - automatically assigning port number, 193–194
 - checksumming, 442–443
 - congestion control, 266
 - connectionless service, 313
 - connection-oriented service, 313–314
 - datagram passed, 337
 - delivering data to socket, 191
 - demultiplexing, 191–198
 - destination host, 191
 - error checking, 203
 - multiplexing, 191–198
 - multiplexing/demultiplexing service, 198–199
 - network layer relationship, 186–189
 - overview, 189–191
 - process-to-process communication, 305, 313
 - reliable data transfer, 204
 - responsibility of delivering data to appropriate application, 191
 - segments, 189
 - services, 186
- transport-layer multiplexing, 192
- transport-layer packets, 186

transport-layer protocols, 50, 91
 end systems implementation, 186
 IP datagrams, 334
 living in end systems, 188
 logical communication between processes, 186, 188–189
 reliable data transfer, 91
 reliable delivery, 436
 security, 93, 705
 TCP (Transmission Control Protocol), 189
 throughput, 92
 timing, 92–93
 UDP (User Datagram Protocol), 189
 Transport Layer Security. *See* TLS
 transport-layer segments, 54–55, 186
 datagrams, 242
 delivering data to correct socket, 191–198
 fields, 191
 unreliability, 242
 transport mode, 721
 transport protocols
 Internet applications, 96
 services, 189
 SSL (Secure Sockets Layer), 712
 TCP, 51
 UDP, 51
 transport services
 available to applications, 91–93
 connection-oriented service, 94
 provided by Internet, 93–96
 reliable data transfer, 91
 security, 93
 TCP services, 94–95
 throughput, 92
 timing, 92–93
 UDP, 95
 trap messages, 773
 tree-join messages, 404–405
 triangle routing problem, 563
 triple-DES, 710
 truncation attack, 717
 TTL (time-to-live) field, 139–140, 334
 tunneling, 360–361, 561

tunnel mode, 721–722
 twisted-pair copper wire, 19–20, 475
 Twitter, 65, 83, 86
 two-dimensional parity scheme, 441–442
 2G cellular networks architecture, 548–550
 Type, Length, Value approach. *See* TLV approach
 type of service bits. *See* TOS bits

U

UDP checksum, 202–204
 UDPClient.py client program, 158–161
 UDP header, 202
 UDP packet, 258, 346, 595
 UDP ports, 258
 UDP segments, 202–204, 495–497, 613
 UDPServer.py server program, 158, 161, 194
 UDP sockets
 communicating to processes, 158
 creation, 161
 identifying, 194
 port numbers, 193–194
 UDP streaming, 593, 595–596
 UDP (User Datagram Protocol), 51, 93, 189, 387
 checksum, 208, 334
 client-server application, 157
 congestion control, 201, 282
 connection establishment, 200
 connectionless transport, 95, 198–204
 connection state, 200
 datagrams, 189
 delays, 200
 destination port number, 199
 development, 62
 directly talking with IP, 199
 discarding damaged segment, 204
 DNS and, 199–200
 end-to-end principle, 203
 end-to-end throughput, 95
 error checking, 199
 error detection, 202–204
 extending IP’s delivery service, 190

- UDP (User Datagram Protocol)
(continued)
- fairness, 282
 - finer application-level control over data, 199
 - flow control, 252
 - gaps in data, 473
 - handshaking, 199
 - header overhead, 200
 - integrity checking, 190
 - Internet checksum, 442
 - Internet telephony applications, 96
 - multimedia applications, 200–201, 282
 - multiplexing/demultiplexing function, 199
 - network management data, 200
 - no-frills segment-delivery service, 199
 - packet loss, 613
 - passing damaged segment to application, 204
 - real-time applications, 200
 - reliable data transfer, 201
 - RIP routing table updates, 200
 - RTP and, 624
 - segments, 189
 - small packet header overhead, 200
 - socket programming, 157–158
 - transport services, 95
 - unreliability, 95, 190
 - wireless networks, 575–577, 301
- UMTS (Universal Mobile Telecommunications Service)
- 3G standards, 550
 - unchoked, 150
 - uncontrolled flooding, 401
 - undetected bit errors, 440
 - unguided media, 19
 - unicast addresses, 356
 - unicast applications and RTP packets, 624
 - unicast communication and IP addresses, 406
 - unidirectional data transfer, 205
 - Universal Plug and Play. *See* UPnP
- UNIX
- BSD (Berkeley Software Distribution)
 - version, 384
 - nslookup program, 141–142
 - RIP implemented in, 387–388
 - Snort, 742
 - unreliable data transfer, 206
 - unreliable service, 190
 - unshielded twisted pair. *See* UTP
 - UPnP (Universal Plug and Play), 352
 - urgent data pointer field, 235
 - URL field, 104
 - URLs, 99
 - US Department of Defense Advanced Research Projects Agency. *See* DARPA
 - user agents, 119–121, 126–127
 - user-based security, 777
 - user-server interaction and HTTP (HyperText Transfer Protocol), 108–110
 - utilization, 217
 - UTP (unshielded twisted pair), 19–20
- V
- VANET (vehicular ad hoc network), 518
 - variables and TCP connection, 233
 - VC networks, 314–317, 319–320
 - VC (virtual-circuit), 267, 314
 - roots in telephony world, 319
 - terminating, 316
 - vehicular ad hoc network. *See* VANET
 - Verizon, 758
 - FIOS service and PONs (passive optical networks), 15–16
 - version number, 333
 - video, 588–589
 - P2P delivery, 611
 - prefetching, 596–597
 - prerecorded, 591
 - repositioning, 600
 - streaming stored, 593–612
 - timing considerations and tolerance of data loss, 592
 - traversing firewalls and NATs, 596

video conferencing, 83
 video over IP, 592–593
 video stream, 625
virtual-circuit. *See* VC (virtual-circuit)
 virtual local area networks. *See* VLANs
 virtual private networks. *See* VPNs
 viruses, 56, 740
 visited MSC, 574
 visited networks, 557, 570
 visitor location register. *See* VLR
 VLANs (virtual local area networks),
 482–486
 VLAN tag, 484–486
 VLAN trunking, 484–485
 VLR (visitor location register), 570
 voice and video applications, 83
 VoIP (Voice-over-IP), 83
 adaptive playout delay, 615–618
 end-to-end delay, 613–614
 enhancing over best-effort network, 612
 fixed playout delay, 615
 jitter and audio, 614–618
 media packetization delays, 44
 packet loss, 613
 recovering from packet loss, 618–621
 sequence numbers, 615
 timestamps, 615
 wireless systems, 668
 VPNs (virtual private networks), 362
 confidentiality, 720
 end points, 725
 IPSec, 718–720
 IPv4, 719
 MPLS (Multiprotocol Label
 Switching), 489–490
 SA (security association), 720
 tunnel mode, 721
 vulnerability attacks, 57

W

Web, 64, 86, 97
 client-server application architecture,
 100
 HTTP (HyperText Transfer Protocol),
 98–100

network applications, 98–116
 operating on demand, 98
 platform for applications emerging
 after 2003, 98
 terminology, 98–99
 Web applications, 97
 client and server processes, 88
 client-server architecture, 86
 Web-based e-mail, 86, 129–130
 Web browsers, 97
 client side of HTTP, 99
 GUI interfaces, 64
 Web caches, 59, 110–115
 Web client-server interaction, 499
 web of trust, 710
 Web pages, 99
 displaying, 101
 requests, 495–500
 Web proxy caches, 104
 Web servers, 89, 97
 deleting objects, 105
 initial versions, 64
 IP addresses, 392
 port numbers, 197–198
 server processes, 88
 server side of HTTP, 99
 spawning new process for connections,
 198
 TCP (Transmission Control Protocol),
 197–198
 uploading objects to, 105
 Web sites, 108
 anonymity, 738
 privacy, 738
 weighted fair queuing. *See* WFQ
 well-known port number, 192
 WEP (Wired Equivalent Privacy), 726–728
 WFQ (weighted fair queuing), 329, 644–645
 leaky bucket, 647–648
 wide-area wireless access, 18
 WiFi, 17, 52, 526–546
 high-speed, 65
 home networks, 17
 hotspots, 515, 546
 public access, 515

- WiMAX (World Interoperability for Microwave Access), 554, 668
- Windows
 nslookup program, 141–142
 Snort, 742
 Wireshark packet sniffer, 78
 window size, 220
- wired-access ISPs tiered levels of service, 636
- wired broadcast links, 521
- wired environments and packet sniffer, 58–59
- Wired Equivalent Privacy. *See WEP*
- wired link differences from wireless links, 519
- wired networks, 519
- wireless, 513–514
- wireless communication links, 515–516
- wireless devices, 58–59
- wireless hosts, 514, 516–517, 530
- wireless LANs, 445
 access point, 17
 LAN base stations, 548
 DHCP (Dynamic Host Configuration Protocol), 346
versus 3G cellular mobile systems, 548
 IEEE 802.11 technology, 17
 security, 726–731
 WiFi, 17
- wireless LANs and 802.11 standards, 526
- wireless links
 bit errors, 519
 decreasing signal strength, 519
 differences from wired links, 519
 fading signal’s strength, 521–522
 hidden terminal problem, 521
 interference from other sources, 519
 multipath propagation, 519
 TCP sender awareness, 577
 undetectable collisions, 521–522
- wireless mesh networks, 518
- wireless networks, 513
 application layer, 575
 base station, 516–518
- CDMA (code division multiple access)
 protocol, 522–526
 characteristics, 519–526
 802.11 wireless LANs, 526–546
 link layer, 575
 link rates, 515
 mobility, 575–577
 multi-hop, infrastructure-based, 518
 multi-hop, infrastructure-less, 518
 network infrastructure, 518
 network layer, 575
 single-hop, infrastructure-based, 518
 single-hop, infrastructure-less, 518
 TCP (Transmission Control Protocol), 575–577
 UDP (User Datagram Protocol), 575–577
 wireless communication links, 515–516
 wireless hosts, 514
- wireless personal area network. *See WPAN*
- Wireless Philadelphia, 515
- wireless station, 529–530
- Wireshark labs, 59, 78
- work-conserving round robin discipline, 644
- workload model, 635
- World Wide Web. *See Web*
- worms, 56–57, 740
- WPAN (wireless personal area network), 544
- X**
- X.25, 512
- XNS (Xerox Network Systems)
 architecture, 384
- Y**
- Yahoo!, 65, 86, 130
- YouTube, 65, 588, 610–611
 HTTP streaming (over TCP), 596
 streaming stored video, 591
 video, 602
- Z**
- Zigbee, 545–546