

KUROSE
•
ROSS

Computer Networking

A TOP-DOWN APPROACH

SEVENTH EDITION

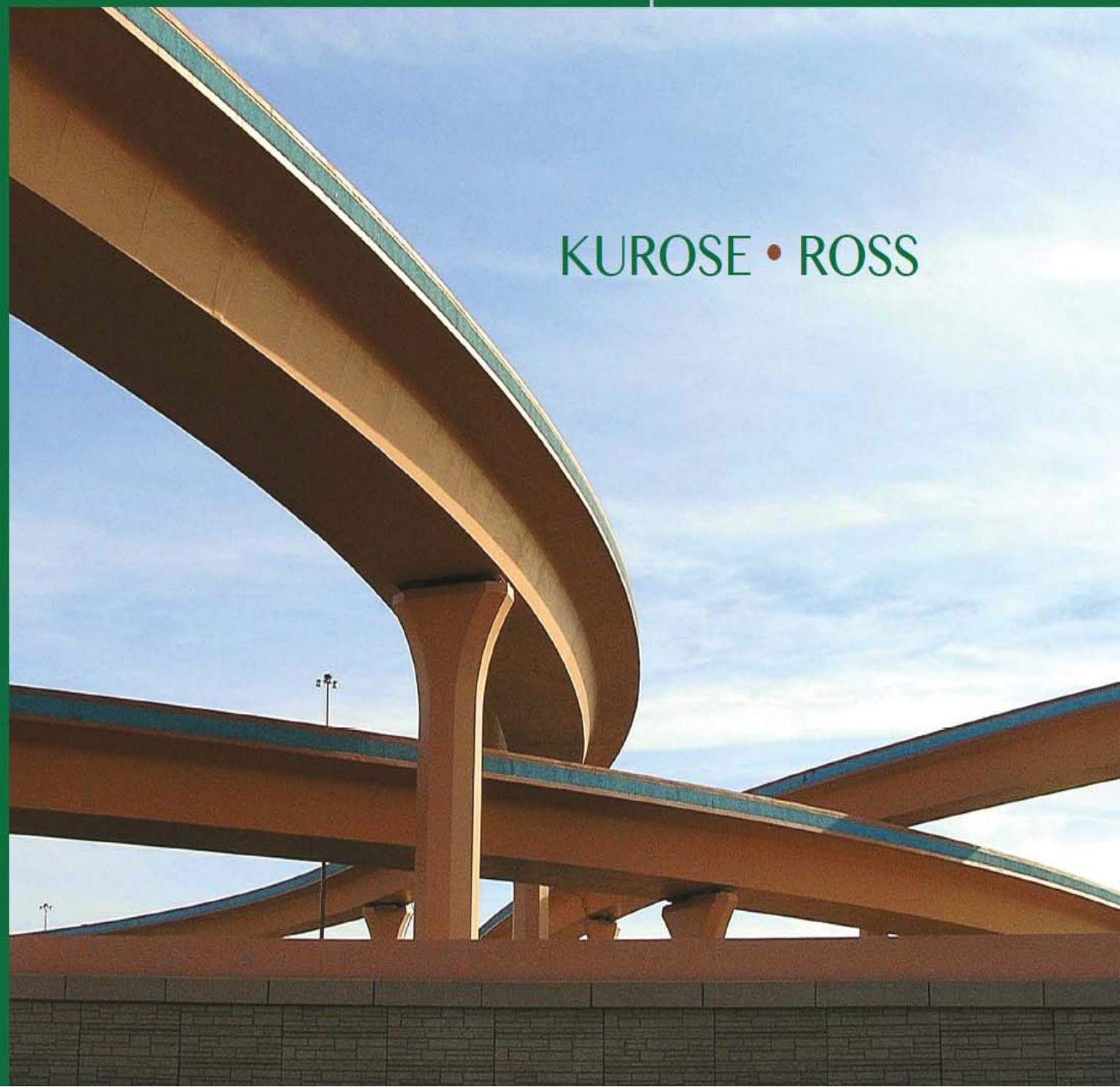
PEARSON

Computer Networking

A TOP-DOWN APPROACH

SEVENTH EDITION

KUROSE • ROSS



Computer Networking

A Top-Down Approach

Seventh Edition

James F. Kurose

University of Massachusetts, Amherst

Keith W. Ross

NYU and NYU Shanghai

PEARSON

Boston Columbus Indianapolis New York San Francisco Hoboken 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, Editorial Director, ECS: Marcia Horton

Acquisitions Editor: Matt Goldstein

Editorial Assistant: Kristy Alaura

Vice President of Marketing: Christy Lesko

Director of Field Marketing: Tim Galligan

Product Marketing Manager: Bram Van Kempen

Field Marketing Manager: Demetrius Hall

Marketing Assistant: Jon Bryant

Director of Product Management: Erin Gregg

Team Lead, Program and Project Management: Scott Disanno

Program Manager: Joanne Manning and Carole Snyder

Project Manager: Katrina Ostler, Ostler Editorial, Inc.

Senior Specialist, Program Planning and Support: Maura Zaldivar-Garcia

Cover Designer: Joyce Wells

Manager, Rights and Permissions: Ben Ferrini

Project Manager, Rights and Permissions: Jenny Hoffman, Aptara Corporation

Inventory Manager: Ann Lam

Cover Image: Marc Gutierrez/Getty Images

Media Project Manager: Steve Wright

Composition: Cenveo Publishing Services

Printer/Binder: Edwards Brothers Malloy

Cover and Insert Printer: Phoenix Color/ Hagerstown

Credits and acknowledgments borrowed from other sources and reproduced, with permission, in this textbook appear on appropriate page within text.

Copyright © 2017, 2013, 2010 **Pearson Education, Inc.** 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. For information regarding permissions, request forms and the appropriate contacts within the Pearson Education Global Rights & Permissions Department, please visit www.pearsoned.com/permissions/. Many of the designations by manufacturers and seller 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

Names: Kurose, James F. | Ross, Keith W., 1956-

Title: Computer networking: a top-down approach / James F. Kurose, University of Massachusetts, Amherst, Keith W. Ross, NYU and NYU Shanghai.

Description: Seventh edition. | Hoboken, New Jersey: Pearson, [2017] | Includes bibliographical references and index.

Identifiers: LCCN 2016004976 | ISBN 9780133594140 | ISBN 0133594149

Subjects: LCSH: Internet. | Computer networks.

Classification: LCC TK5105.875.I57 K88 2017 | DDC 004.6-dc23



ISBN-10: 0-13-359414-9

ISBN-13: 978-0-13-359414-0

About the Authors

Jim Kurose

Jim Kurose is a Distinguished University Professor of Computer Science at the University of Massachusetts, Amherst. He is currently on leave from the University of Massachusetts, serving as an Assistant Director at the US National Science Foundation, where he leads the Directorate of Computer and Information Science and Engineering.

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 won several conference best paper awards and received the IEEE Infocom Achievement Award and the ACM Sigcomm Test of Time Award.



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

Keith Ross

Keith Ross is the Dean of Engineering and Computer Science at NYU Shanghai and the Leonard J. Shustek Chair Professor in the Computer Science and Engineering Department at NYU. Previously he was at University of Pennsylvania (13 years), Eurecom Institute (5 years) and Polytechnic University (10 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 co-founder and original CEO of Wimba, which develops online multimedia applications for e-learning and was acquired by Blackboard in 2010.*



Professor Ross's research interests are in privacy, social networks, peer-to-peer networking, Internet measurement, content distribution networks, and stochastic modeling. He is an ACM Fellow, 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.

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

Preface

Welcome to the seventh edition of *Computer Networking: A Top-Down Approach*. Since the publication of the first edition 16 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 Seventh 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 seventh 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 seventh edition has been revised and updated substantially.

Long-time readers of our book will notice that for the first time since this text was published, we've changed the organization of the chapters themselves. The network layer, which had been previously covered in a single chapter, is now covered in [Chapter 4](#) (which focuses on the so-called “data plane” component of the network layer) and [Chapter 5](#) (which focuses on the network layer’s “control plane”). This expanded coverage of the network layer reflects the swift rise in importance of software-defined networking (SDN), arguably the most important and exciting advance in networking in decades. Although a relatively recent innovation, SDN has been rapidly adopted in practice—so much so that it's already hard to imagine an introduction to modern computer networking that doesn't cover SDN. The topic of network management, previously covered in [Chapter 9](#), has now been folded into the new [Chapter 5](#). As always, we've also updated many other sections of the text to reflect recent changes in the dynamic field of networking since the sixth edition. As always, material that has been retired from the printed text can always be found on this book's Companion Website. The most important updates are the following:

- [Chapter 1](#) has been updated to reflect the ever-growing reach and use of the Internet.
- [Chapter 2](#), which covers the application layer, has been significantly updated. We've removed the material on the FTP protocol and distributed hash tables to make room for a new section on application-level video streaming and content distribution networks, together with Netflix and YouTube case studies. The socket programming sections have been updated from Python 2 to Python 3.
- [Chapter 3](#), which covers the transport layer, has been modestly updated. The material on asynchronous transport mode (ATM) networks has been replaced by more modern material on the Internet's explicit congestion notification (ECN), which teaches the same principles.
- [Chapter 4](#) covers the “data plane” component of the network layer—the *per-router* forwarding function that determine how a packet arriving on one of a router's input links is forwarded to one of that router's output links. We updated the material on traditional Internet forwarding found in all previous editions, and added material on packet scheduling. We've also added a new section on generalized forwarding, as practiced in SDN. There are also numerous updates throughout the chapter. Material on multicast and broadcast communication has been removed to make way for the new material.
- In [Chapter 5](#), we cover the control plane functions of the network layer—the *network-wide* logic that controls how a datagram is routed along an end-to-end path of routers from the source host to the destination host. As in previous editions, we cover routing algorithms, as well as routing protocols (with an updated treatment of BGP) used in today's Internet. We've added a significant new section on the SDN control plane, where routing and other functions are implemented in so-called SDN controllers.
- [Chapter 6](#), which now covers the link layer, has an updated treatment of Ethernet, and of data center networking.
- [Chapter 7](#), which covers wireless and mobile networking, contains updated material on 802.11 (so-called “WiFi) networks and cellular networks, including 4G and LTE.
- [Chapter 8](#), which covers network security and was extensively updated in the sixth edition, has only

modest updates in this seventh edition.

- **Chapter 9**, on multimedia networking, is now slightly “thinner” than in the sixth edition, as material on video streaming and content distribution networks has been moved to **Chapter 2**, and material on packet scheduling has been incorporated into **Chapter 4**.
- Significant new material involving end-of-chapter problems has been added. As with all previous editions, homework problems have been revised, added, and removed.

As always, our aim in creating this new edition of our book is to continue to provide a focused and modern treatment of computer networking, emphasizing both principles and practice.

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 16 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. Three fundamentally important network-layer issues are determining “good” paths between two routers, interconnecting a large number of heterogeneous networks, and managing the complexity of a modern network. 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 Website

Each new copy of this textbook includes twelve months of access to a Companion Website for all book readers at <http://www.pearsonhighered.com/cs-resources/>, which includes:

- **Interactive learning material.** The book's Companion Website contains VideoNotes—video presentations of important topics throughout the book done by the authors, as well as walkthroughs of solutions to problems similar to those at the end of the chapter. 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 FTP, distributed hash tables, and multicasting. Material that appeared in earlier editions of the text is still of interest, and thus 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.

In addition to the Companion Website, the authors maintain a public Web site, http://gaia.cs.umass.edu/kurose_ross/interactive, containing interactive exercises that 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.

Pedagogical Features

We have each been teaching computer networking for more than 30 years. Together, we bring more than 60 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 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 seventh 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 six 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 6** in sequence, following our top-down philosophy. Each of these five chapters leverages material from the preceding chapters. After completing the first six chapters, the instructor has quite a bit of flexibility. There are no interdependencies among the last three chapters, so they can be taught in any order. However, each of the last three chapters depends on the material in the first six chapters. Many instructors first teach the first six chapters and then teach one of the last three 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 five 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 keithross@nyu.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 seventh 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)

David Fullager (Netflix)

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)

Dan Rubenstein (Columbia University)

Avi Rubin (Johns Hopkins 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)

Dian Yu (NYU Shanghai)

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)

Mike Zink (University of Massachusetts)

Cliff C. Zou (University of Central Florida)

We also want to thank the entire Pearson team—in particular, Matt Goldstein and Joanne Manning—who have done an absolutely outstanding job on this seventh 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 and earlier editions of our book, and to Katie Ostler and her team at Cenveo for their wonderful production work on this edition. Finally, a most special thanks go to our previous two editors at Addison-Wesley—Michael Hirsch and Susan Hartman. 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 21

1.3.1 Packet Switching 23

1.3.2 Circuit Switching 27

1.3.3 A Network of Networks 31

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 41

1.4.4 Throughput in Computer Networks 43

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 59

1.7.1 The Development of Packet Switching: 1961–1972 59

1.7.2 Proprietary Networks and Internetworking: 1972–1980 60

1.7.3 A Proliferation of Networks: 1980–1990 62

1.7.4 The Internet Explosion: The 1990s 63

1.7.5 The New Millennium 64

1.8 Summary 65

Homework Problems and Questions 67

Wireshark Lab 77

Interview: Leonard Kleinrock 79

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 90

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

2.3 Electronic Mail in the Internet 116

2.3.1 SMTP 118

2.3.2 Comparison with HTTP 121

2.3.3 Mail Message Formats 121

2.3.4 Mail Access Protocols 122

2.4 DNS—The Internet’s Directory Service 126

2.4.1 Services Provided by DNS 127

2.4.2 Overview of How DNS Works 129

2.4.3 DNS Records and Messages 135

2.5 Peer-to-Peer Applications 140

2.5.1 P2P File Distribution 140

2.6 Video Streaming and Content Distribution Networks 147

2.6.1 Internet Video 148

2.6.2 HTTP Streaming and DASH 148

2.6.3 Content Distribution Networks	149
2.6.4 Case Studies: Netflix, YouTube, and Kankan	153
2.7 Socket Programming: Creating Network Applications	157
2.7.1 Socket Programming with UDP	159
2.7.2 Socket Programming with TCP	164
2.8 Summary	170
Homework Problems and Questions	171
Socket Programming Assignments	180
Wireshark Labs: HTTP, DNS	182
Interview: Marc Andreessen	184
Chapter 3 Transport Layer	187
3.1 Introduction and Transport-Layer Services	188
3.1.1 Relationship Between Transport and Network Layers	188
3.1.2 Overview of the Transport Layer in the Internet	191
3.2 Multiplexing and Demultiplexing	193
3.3 Connectionless Transport: UDP	200
3.3.1 UDP Segment Structure	204
3.3.2 UDP Checksum	204
3.4 Principles of Reliable Data Transfer	206
3.4.1 Building a Reliable Data Transfer Protocol	208
3.4.2 Pipelined Reliable Data Transfer Protocols	217
3.4.3 Go-Back-N (GBN)	221
3.4.4 Selective Repeat (SR)	226
3.5 Connection-Oriented Transport: TCP	233
3.5.1 The TCP Connection	233
3.5.2 TCP Segment Structure	236
3.5.3 Round-Trip Time Estimation and Timeout	241
3.5.4 Reliable Data Transfer	244
3.5.5 Flow Control	252
3.5.6 TCP Connection Management	255
3.6 Principles of Congestion Control	261

3.6.1 The Causes and the Costs of Congestion 261

3.6.2 Approaches to Congestion Control 268

3.7 TCP Congestion Control 269

3.7.1 Fairness 279

3.7.2 Explicit Congestion Notification (ECN): Network-assisted Congestion Control 282

3.8 Summary 284

Homework Problems and Questions 286

Programming Assignments 301

Wireshark Labs: Exploring TCP, UDP 302

Interview: Van Jacobson 303

Chapter 4 The Network Layer: Data Plane 305

4.1 Overview of Network Layer 306

4.1.1 Forwarding and Routing: The Network Data and Control Planes 306

4.1.2 Network Service Models 311

4.2 What's Inside a Router? 313

4.2.1 Input Port Processing and Destination-Based Forwarding 316

4.2.2 Switching 319

4.2.3 Output Port Processing 321

4.2.4 Where Does Queuing Occur? 321

4.2.5 Packet Scheduling 325

4.3 The Internet Protocol (IP): IPv4, Addressing, IPv6, and More 329

4.3.1 IPv4 Datagram Format 330

4.3.2 IPv4 Datagram Fragmentation 332

4.3.3 IPv4 Addressing 334

4.3.4 Network Address Translation (NAT) 345

4.3.5 IPv6 348

4.4 Generalized Forwarding and SDN 354

4.4.1 Match 356

4.4.2 Action 358

4.4.3 OpenFlow Examples of Match-plus-action in Action 358

4.5 Summary 361

Homework Problems and Questions 361

Wireshark Lab 370

Interview: Vinton G. Cerf 371

Chapter 5 The Network Layer: Control Plane 373

5.1 Introduction 374

5.2 Routing Algorithms 376

5.2.1 The Link-State (LS) Routing Algorithm 379

5.2.2 The Distance-Vector (DV) Routing Algorithm 384

5.3 Intra-AS Routing in the Internet: OSPF 391

5.4 Routing Among the ISPs: BGP 395

5.4.1 The Role of BGP 395

5.4.2 Advertising BGP Route Information 396

5.4.3 Determining the Best Routes 398

5.4.4 IP-Anycast 402

5.4.5 Routing Policy 403

5.4.6 Putting the Pieces Together: Obtaining Internet Presence 406

5.5 The SDN Control Plane 407

5.5.1 The SDN Control Plane: SDN Controller and SDN Control Applications 410

5.5.2 OpenFlow Protocol 412

5.5.3 Data and Control Plane Interaction: An Example 414

5.5.4 SDN: Past and Future 415

5.6 ICMP: The Internet Control Message Protocol 419

5.7 Network Management and SNMP 421

5.7.1 The Network Management Framework 422

5.7.2 The Simple Network Management Protocol (SNMP) 424

5.8 Summary 426

Homework Problems and Questions 427

Socket Programming Assignment 433

Programming Assignment 434

Wireshark Lab 435

Interview: Jennifer Rexford 436

Chapter 6 The Link Layer and LANs 439

6.1 Introduction to the Link Layer 440

6.1.1 The Services Provided by the Link Layer 442

6.1.2 Where Is the Link Layer Implemented? 443

6.2 Error-Detection and -Correction Techniques 444

6.2.1 Parity Checks 446

6.2.2 Checksumming Methods 448

6.2.3 Cyclic Redundancy Check (CRC) 449

6.3 Multiple Access Links and Protocols 451

6.3.1 Channel Partitioning Protocols 453

6.3.2 Random Access Protocols 455

6.3.3 Taking-Turns Protocols 464

6.3.4 DOCSIS: The Link-Layer Protocol for Cable Internet Access 465

6.4 Switched Local Area Networks 467

6.4.1 Link-Layer Addressing and ARP 468

6.4.2 Ethernet 474

6.4.3 Link-Layer Switches 481

6.4.4 Virtual Local Area Networks (VLANs) 487

6.5 Link Virtualization: A Network as a Link Layer 491

6.5.1 Multiprotocol Label Switching (MPLS) 492

6.6 Data Center Networking 495

6.7 Retrospective: A Day in the Life of a Web Page Request 500

6.7.1 Getting Started: DHCP, UDP, IP, and Ethernet 500

6.7.2 Still Getting Started: DNS and ARP 502

6.7.3 Still Getting Started: Intra-Domain Routing to the DNS Server 503

6.7.4 Web Client-Server Interaction: TCP and HTTP 504

6.8 Summary 506

Homework Problems and Questions 507

Wireshark Lab 515

Interview: Simon S. Lam 516

Chapter 7 Wireless and Mobile Networks 519

7.1 Introduction 520

7.2 Wireless Links and Network Characteristics 525

7.2.1 CDMA 528

7.3 WiFi: 802.11 Wireless LANs 532

7.3.1 The 802.11 Architecture 533

7.3.2 The 802.11 MAC Protocol 537

7.3.3 The IEEE 802.11 Frame 542

7.3.4 Mobility in the Same IP Subnet 546

7.3.5 Advanced Features in 802.11 547

7.3.6 Personal Area Networks: Bluetooth and Zigbee 548

7.4 Cellular Internet Access 551

7.4.1 An Overview of Cellular Network Architecture 551

7.4.2 3G Cellular Data Networks: Extending the Internet to Cellular Subscribers 554

7.4.3 On to 4G: LTE 557

7.5 Mobility Management: Principles 560

7.5.1 Addressing 562

7.5.2 Routing to a Mobile Node 564

7.6 Mobile IP 570

7.7 Managing Mobility in Cellular Networks 574

7.7.1 Routing Calls to a Mobile User 576

7.7.2 Handoffs in GSM 577

7.8 Wireless and Mobility: Impact on Higher-Layer Protocols 580

7.9 Summary 582

Homework Problems and Questions 583

Wireshark Lab 588

Interview: Deborah Estrin 589

Chapter 8 Security in Computer Networks 593

8.1 What Is Network Security? 594

8.2 Principles of Cryptography 596

8.2.1 Symmetric Key Cryptography 598

8.2.2 Public Key Encryption 604

8.3 Message Integrity and Digital Signatures 610

8.3.1 Cryptographic Hash Functions 611

8.3.2 Message Authentication Code 613

8.3.3 Digital Signatures 614

8.4 End-Point Authentication 621

8.4.1 Authentication Protocol ap1.0 622

8.4.2 Authentication Protocol ap2.0 622

8.4.3 Authentication Protocol ap3.0 623

8.4.4 Authentication Protocol ap3.1 623

8.4.5 Authentication Protocol ap4.0 624

8.5 Securing E-Mail 626

8.5.1 Secure E-Mail 627

8.5.2 PGP 630

8.6 Securing TCP Connections: SSL 631

8.6.1 The Big Picture 632

8.6.2 A More Complete Picture 635

8.7 Network-Layer Security: IPsec and Virtual Private Networks 637

8.7.1 IPsec and Virtual Private Networks (VPNs) 638

8.7.2 The AH and ESP Protocols 640

8.7.3 Security Associations 640

8.7.4 The IPsec Datagram 641

8.7.5 IKE: Key Management in IPsec 645

8.8 Securing Wireless LANs 646

8.8.1 Wired Equivalent Privacy (WEP) 646

8.8.2 IEEE 802.11i 648

8.9 Operational Security: Firewalls and Intrusion Detection Systems 651

8.9.1 Firewalls 651

8.9.2 Intrusion Detection Systems 659

8.10 Summary 662

Homework Problems and Questions 664

Wireshark Lab 672

IPsec Lab 672

Interview: Steven M. Bellovin 673

Chapter 9 Multimedia Networking 675

9.1 Multimedia Networking Applications 676

9.1.1 Properties of Video 676

9.1.2 Properties of Audio 677

9.1.3 Types of Multimedia Network Applications 679

9.2 Streaming Stored Video 681

9.2.1 UDP Streaming 683

9.2.2 HTTP Streaming 684

9.3 Voice-over-IP 688

9.3.1 Limitations of the Best-Effort IP Service 688

9.3.2 Removing Jitter at the Receiver for Audio 691

9.3.3 Recovering from Packet Loss 694

9.3.4 Case Study: VoIP with Skype 697

9.4 Protocols for Real-Time Conversational Applications 700

9.4.1 RTP 700

9.4.2 SIP 703

9.5 Network Support for Multimedia 709

9.5.1 Dimensioning Best-Effort Networks 711

9.5.2 Providing Multiple Classes of Service 712

9.5.3 Diffserv 719

9.5.4 Per-Connection Quality-of-Service (QoS) Guarantees: Resource Reservation and Call Admission 723

9.6 Summary 726

Homework Problems and Questions 727

Programming Assignment 735

Interview: Henning Schulzrinne 736

References 741

Index 783

Chapter 1 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 "things" including game consoles, surveillance systems, watches, eye glasses, thermostats, body scales, and cars. 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 answer 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

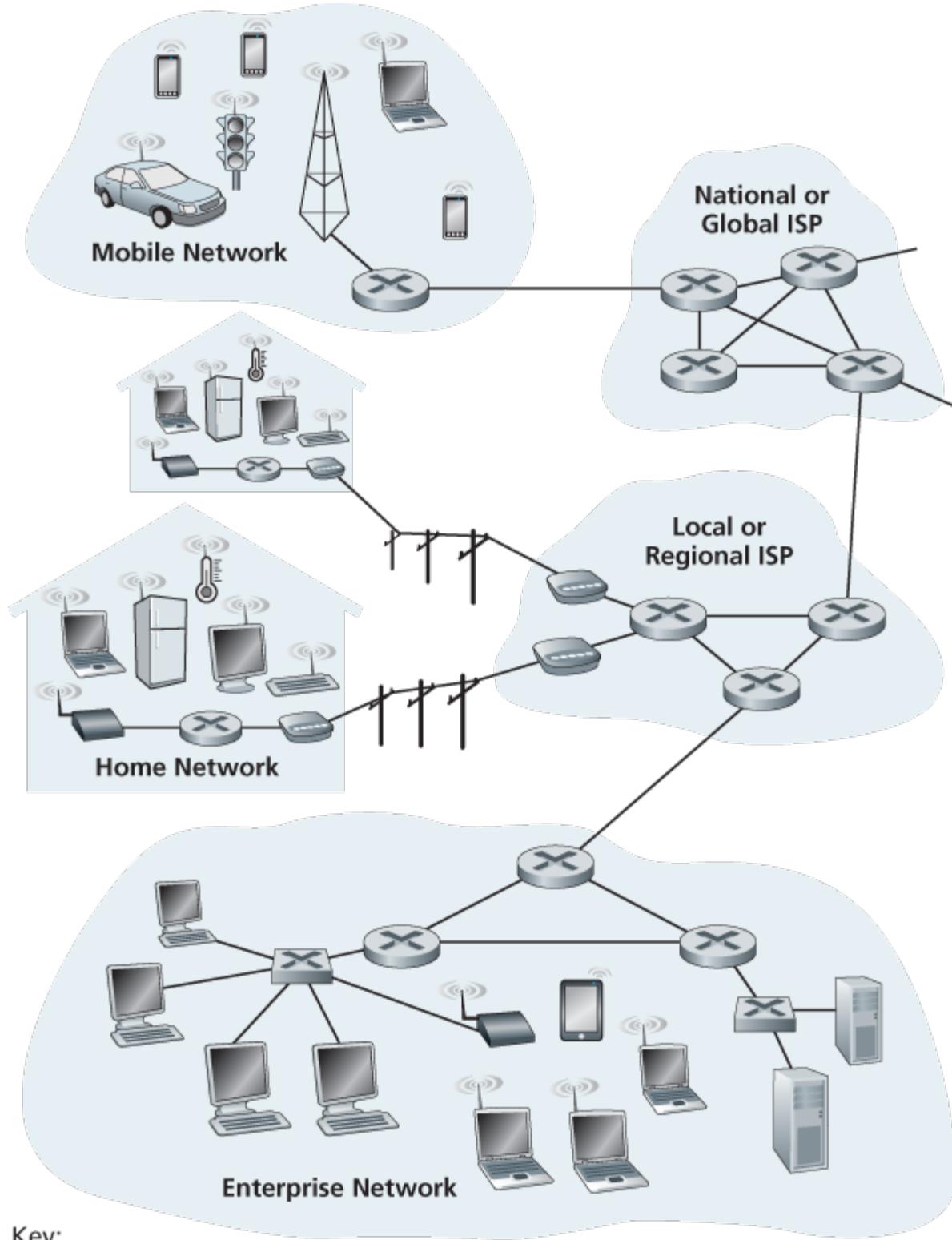
some of these attacks and consider how computer networks can be made more secure. Finally, we'll close this chapter with a brief history of computer networking.

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 billions 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 “things” such as laptops, smartphones, tablets, TVs, gaming consoles, thermostats, home security systems, home appliances, watches, eye glasses, cars, traffic control systems and more 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**. By some estimates, in 2015 there were about 5 billion devices connected to the Internet, and the number will reach 25 billion by 2020 [\[Gartner 2014\]](#). It is estimated that in 2015 there were over 3.2 billion Internet users worldwide, approximately 40% of the world population [\[ITU 2015\]](#).



Key:

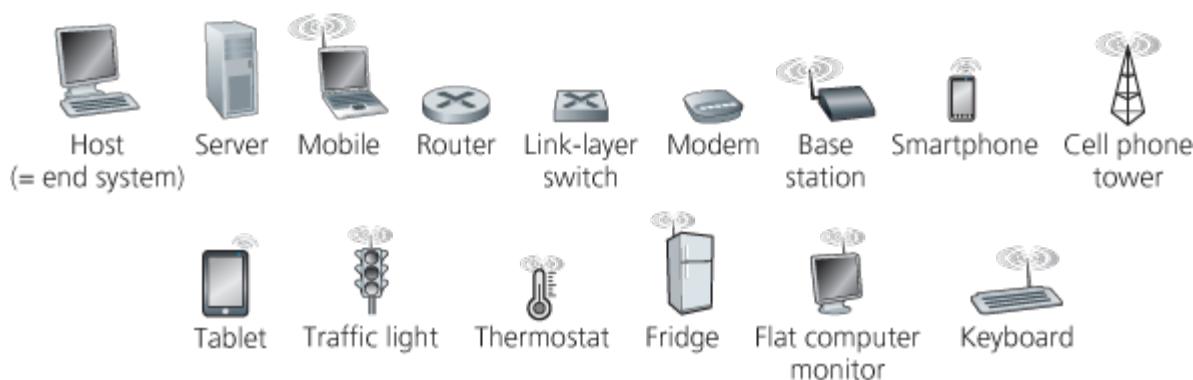


Figure 1.1 Some pieces of the Internet

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. Cisco predicts annual global IP traffic will pass the zettabyte (10^{21} bytes) threshold by the end of 2016, and will reach 2 zettabytes per year by 2019 [Cisco VNI 2015].

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; ISPs that provide WiFi access in airports, hotels, coffee shops, and other public places; and cellular data ISPs, providing mobile access to our smartphones and other devices. 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, and mobile wireless access. ISPs also provide Internet access to content providers, connecting Web sites and video servers 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) [\[IETF 2016\]](#). 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 7,000 RFCs. Other bodies also specify standards for network components, most notably for network links. The IEEE 802 LAN/MAN Standards Committee [\[IEEE 802 2016\]](#), 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*. In addition to traditional applications such as e-mail and Web surfing, Internet applications include mobile smartphone and tablet applications, including Internet messaging, mapping with real-time road-traffic information, music streaming from the cloud, movie and television streaming, online social networks, video conferencing, multi-person games, and location-based recommendation systems. 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 a **socket interface** 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 socket interface 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 socket interface 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 interface," 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 a socket interface 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 thermostat or body scale 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

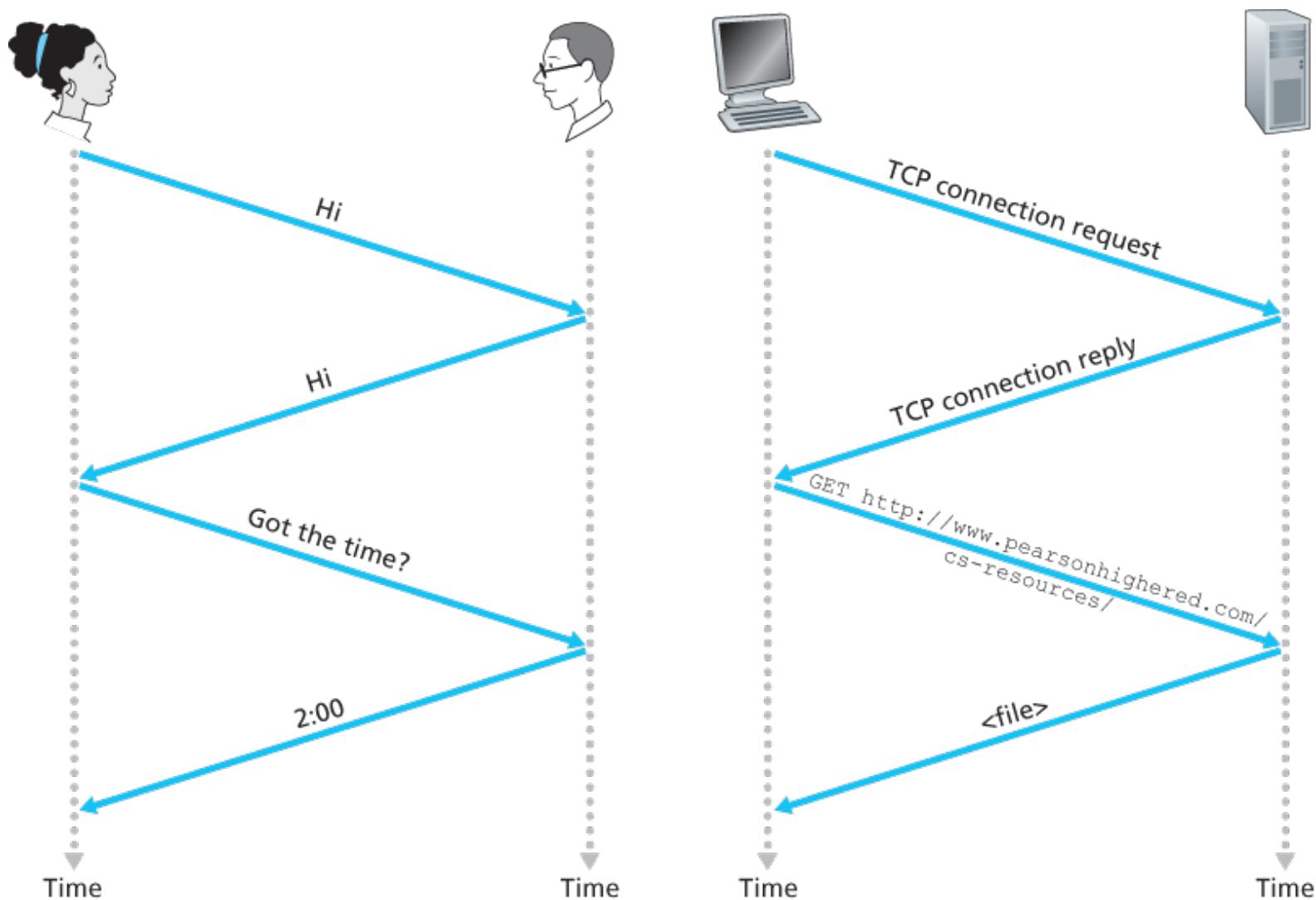


Figure 1.2 A human protocol and a computer network protocol

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 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.

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 devices (e.g., laptops, smartphones, and tablets). Furthermore, an increasing number of non-traditional “things” are being attached to the Internet as end systems (see the Case History feature).

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

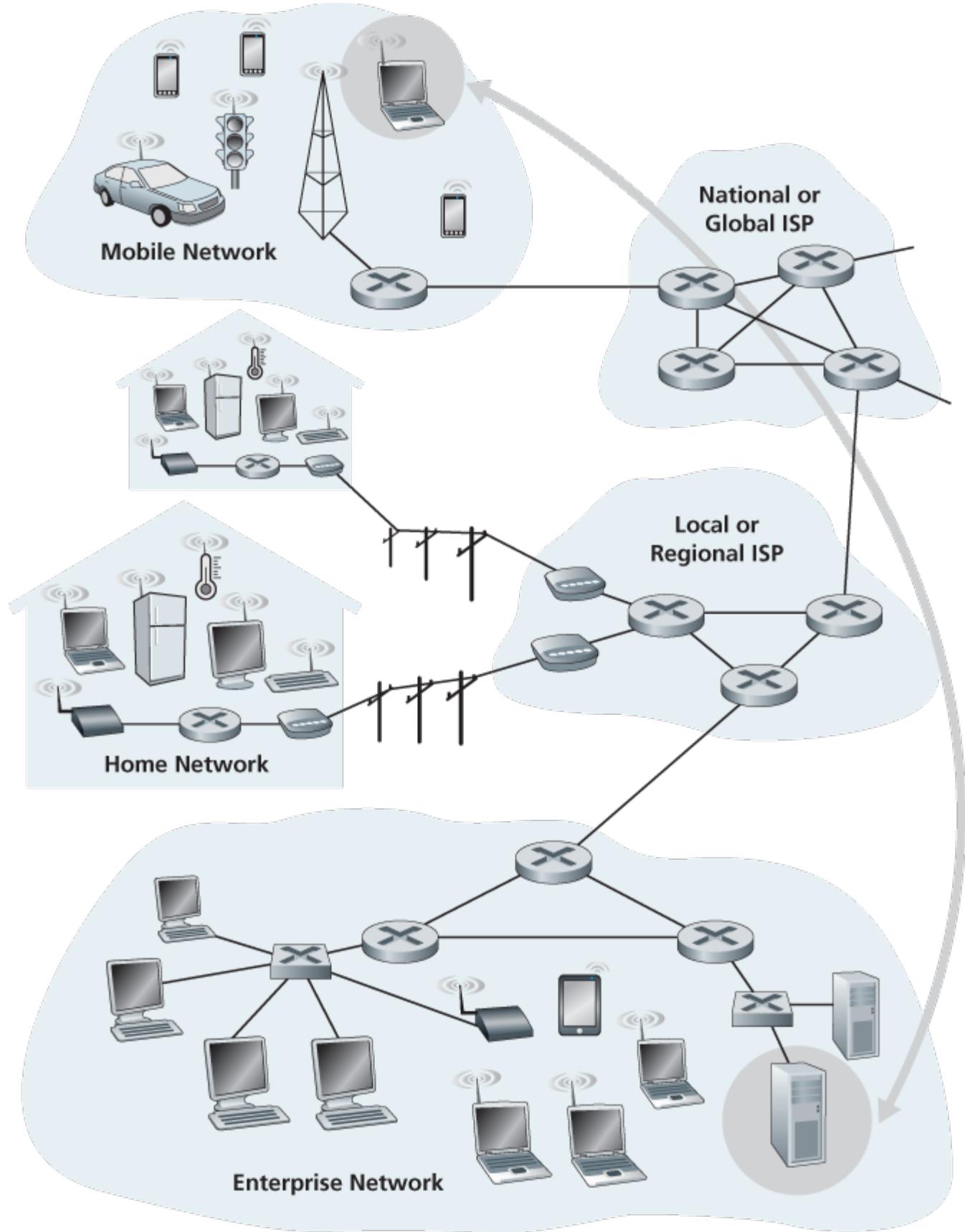


Figure 1.3 End-system interaction

CASE HISTORY

THE INTERNET OF THINGS

Can you imagine a world in which just about everything is wirelessly connected to the Internet? A world in which most people, cars, bicycles, eye glasses, watches, toys, hospital equipment, home sensors, classrooms, video surveillance systems, atmospheric sensors, store-shelf

products, and pets are connected? This world of the Internet of Things (IoT) may actually be just around the corner.

By some estimates, as of 2015 there are already 5 billion things connected to the Internet, and the number could reach 25 billion by 2020 [[Gartner 2014](#)]. These things include our smartphones, which already follow us around in our homes, offices, and cars, reporting our geo-locations and usage data to our ISPs and Internet applications. But in addition to our smartphones, a wide-variety of non-traditional “things” are already available as products. For example, there are Internet-connected wearables, including watches (from Apple and many others) and eye glasses. Internet-connected glasses can, for example, upload everything we see to the cloud, allowing us to share our visual experiences with people around the world in real-time. There are Internet-connected things already available for the smart home, including Internet-connected thermostats that can be controlled remotely from our smartphones, and Internet-connected body scales, enabling us to graphically review the progress of our diets from our smartphones. There are Internet-connected toys, including dolls that recognize and interpret a child’s speech and respond appropriately.

The IoT offers potentially revolutionary benefits to users. But at the same time there are also huge security and privacy risks. For example, attackers, via the Internet, might be able to hack into IoT devices or into the servers collecting data from IoT devices. For example, an attacker could hijack an Internet-connected doll and talk directly with a child; or an attacker could hack into a database that stores personal health and activity information collected from wearable devices. These security and privacy concerns could undermine the consumer confidence necessary for the technologies to meet their full potential and may result in less widespread adoption [[FTC 2015](#)].

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 search results, e-mail, Web pages, and videos reside in large **data centers**. For example, Google has 50-100 data centers, including about 15 large centers, each with more than 100,000 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

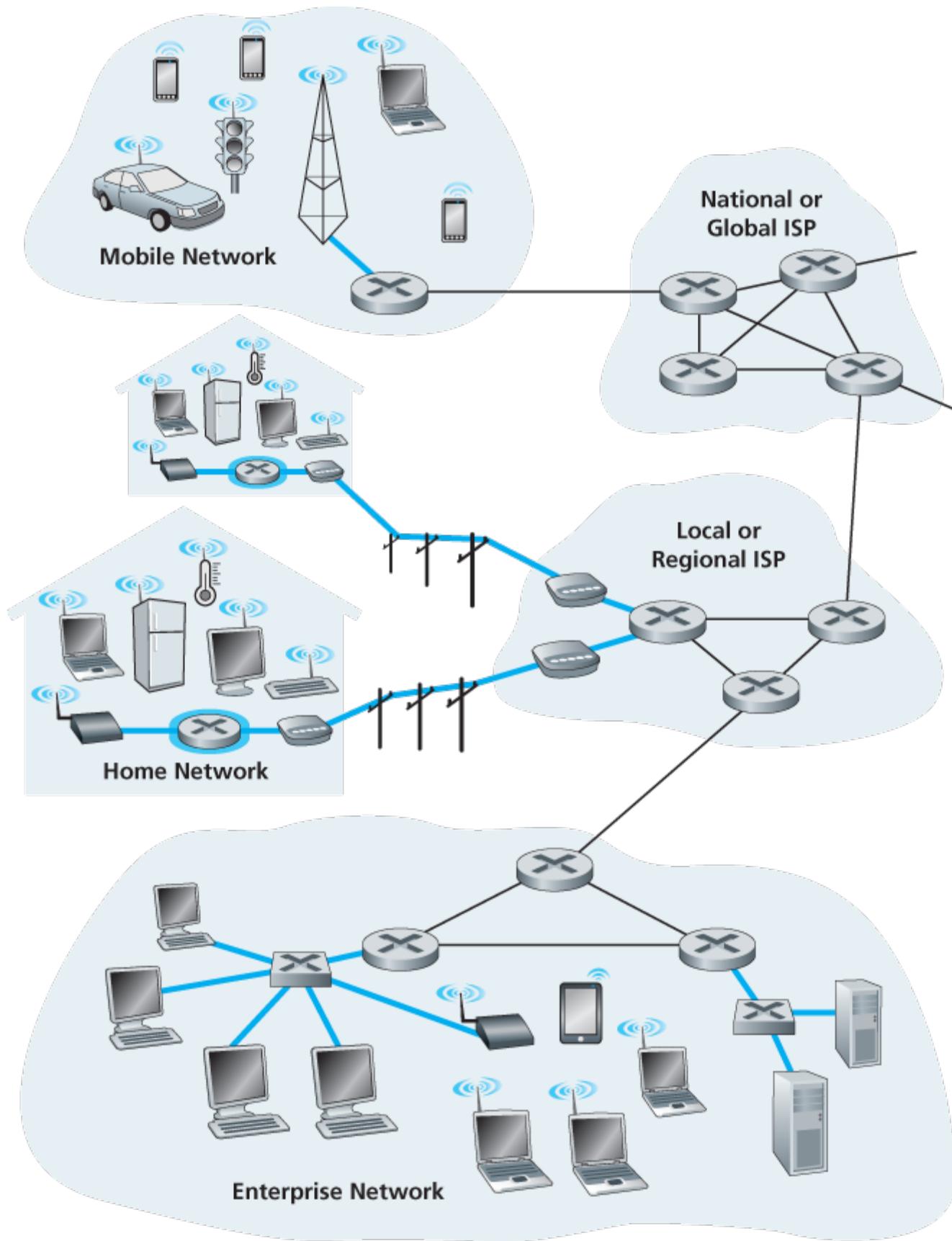


Figure 1.4 Access networks

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 as of 2014, more than 78 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 2015]. Given this widespread use of home access networks 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.

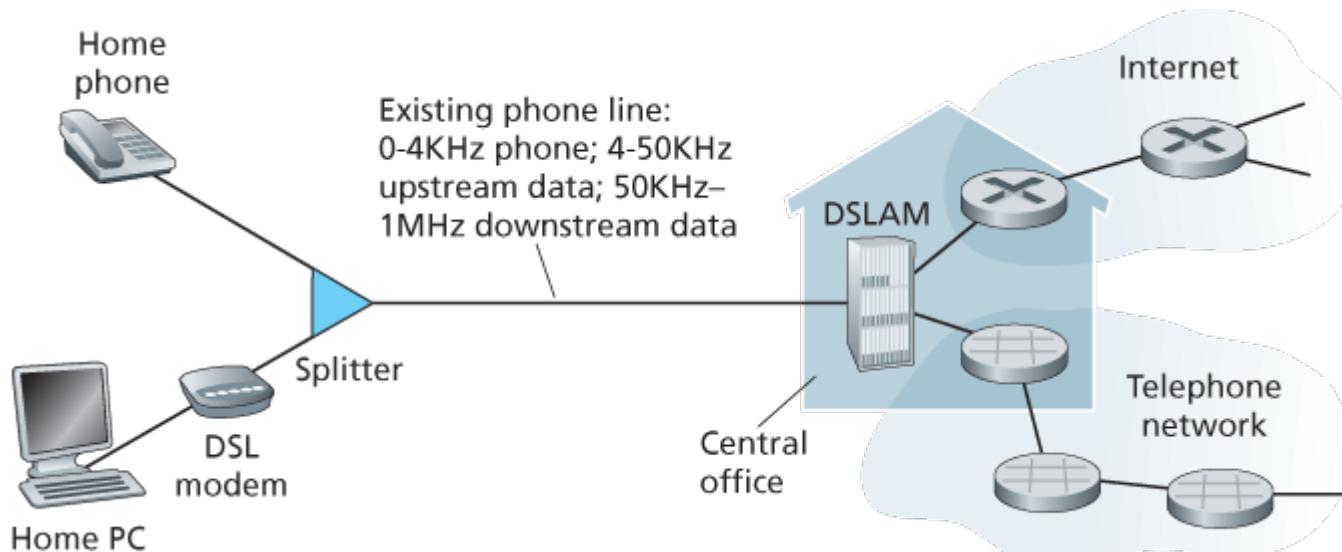


Figure 1.5 DSL Internet access

(We'll describe this technique of frequency-division multiplexing in **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 multiple transmission rates, including 12 Mbps downstream and 1.8 Mbps upstream [ITU 1999], and 55 Mbps downstream and 15 Mbps upstream [ITU 2006]. 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. The maximum rate is also 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).

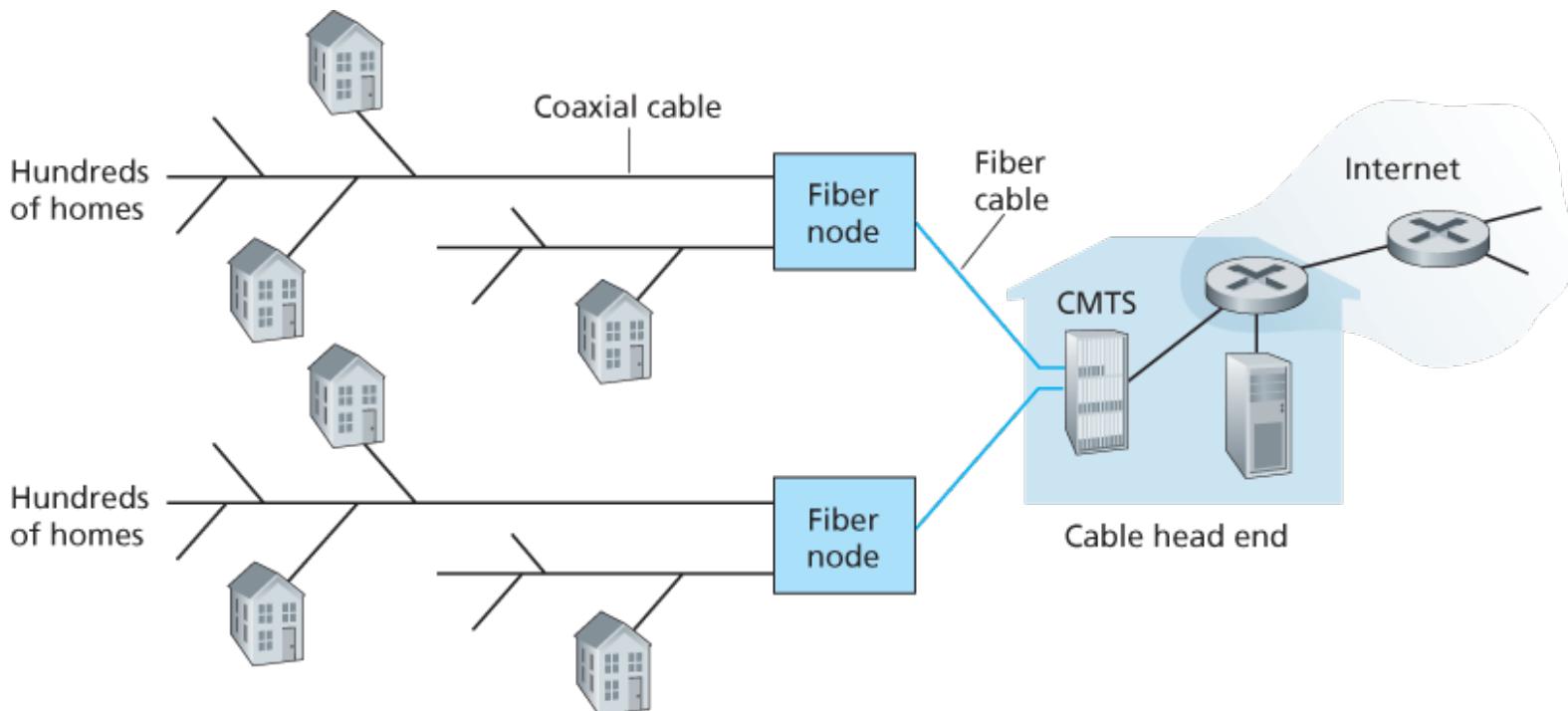


Figure 1.6 A hybrid fiber-coaxial access network

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 6](#).) 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 6](#).)

Although DSL and cable networks currently represent more than 85 percent of residential broadband access in the United States, an up-and-coming technology that provides even higher speeds is [fiber to the home \(FTTH\)](#) [[FTTH Council 2016](#)]. As the name suggests, the FTTH concept is simple—provide an optical fiber path from the CO directly to the home. Many countries today—including the UAE, South Korea, Hong Kong, Japan, Singapore, Taiwan, Lithuania, and Sweden—now have household penetration rates exceeding 30% [[FTTH Council 2016](#)].

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 6](#).

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 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

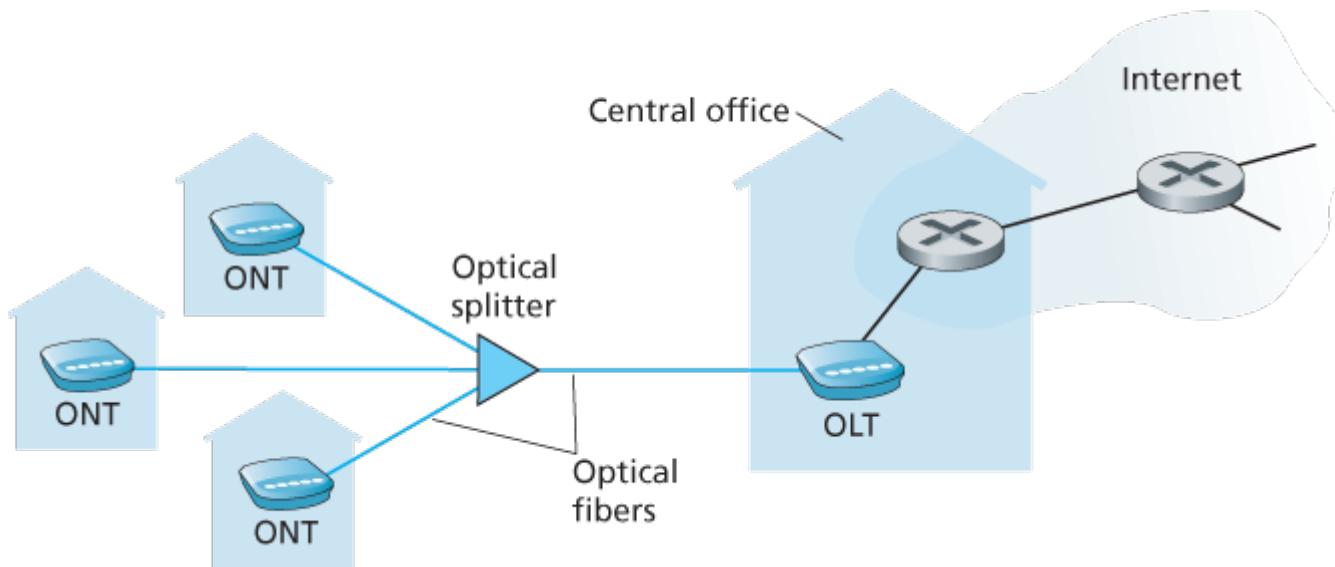


Figure 1.7 FTTH Internet access

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 technology discussed in detail in [Chapter 6](#). The Ethernet switch, or a network of such

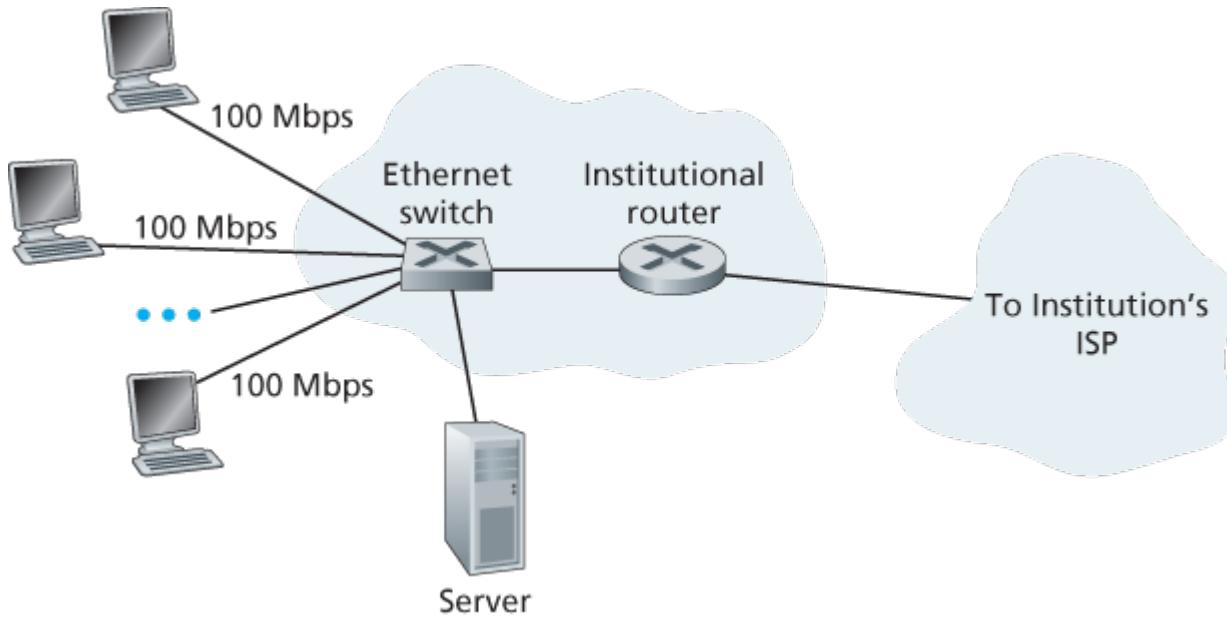


Figure 1.8 Ethernet Internet access

interconnected switches, is then in turn connected into the larger Internet. With Ethernet access, users typically have 100 Mbps or 1 Gbps 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 “things” (see earlier sidebar on “[Internet of Things](#)”). 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 using 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 [\[wigle.net 2016\]](#)). As discussed in detail in [Chapter 7](#), 802.11 today provides a shared transmission rate of up to more than 100 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 and other wireless devices in the home; 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.

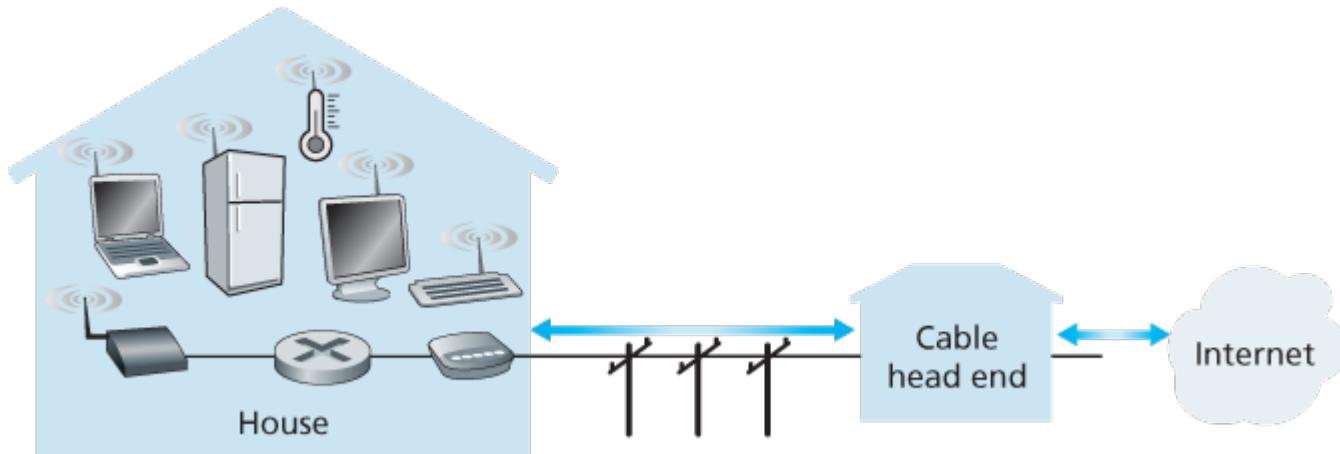


Figure 1.9 A typical home network

Wide-Area Wireless Access: 3G and LTE

Increasingly, devices such as iPhones and Android devices are being used to message, share photos in social networks, watch movies, and stream 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 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 7](#).

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 (or those of our parents or grandparents!) 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 central office).

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. [\[Mukherjee 2006, Ramaswami 2010\]](#) 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 7](#).

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 \[Wiki Satellite 2016\]](#).

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. 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.

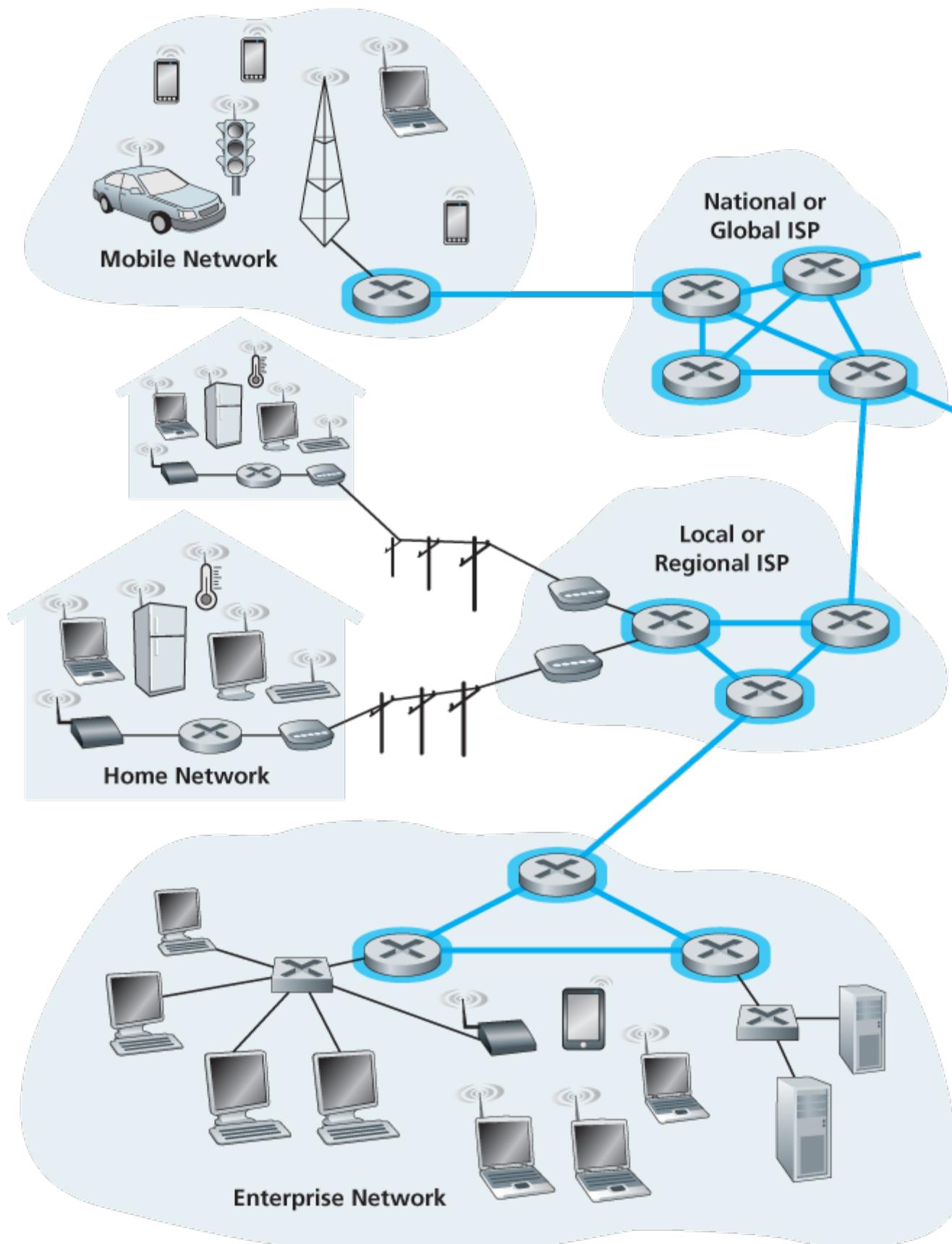


Figure 1.10 The network core

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 e-mail 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 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

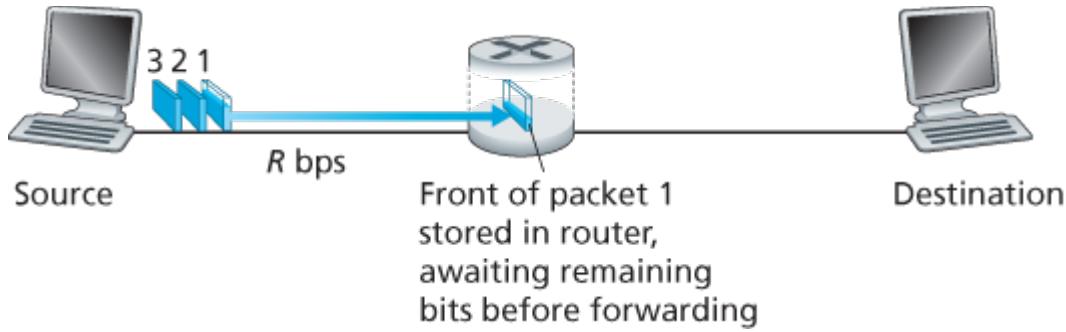


Figure 1.11 Store-and-forward packet switching

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:

$$\text{end-to-end} = NLR \quad (1.1)$$

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

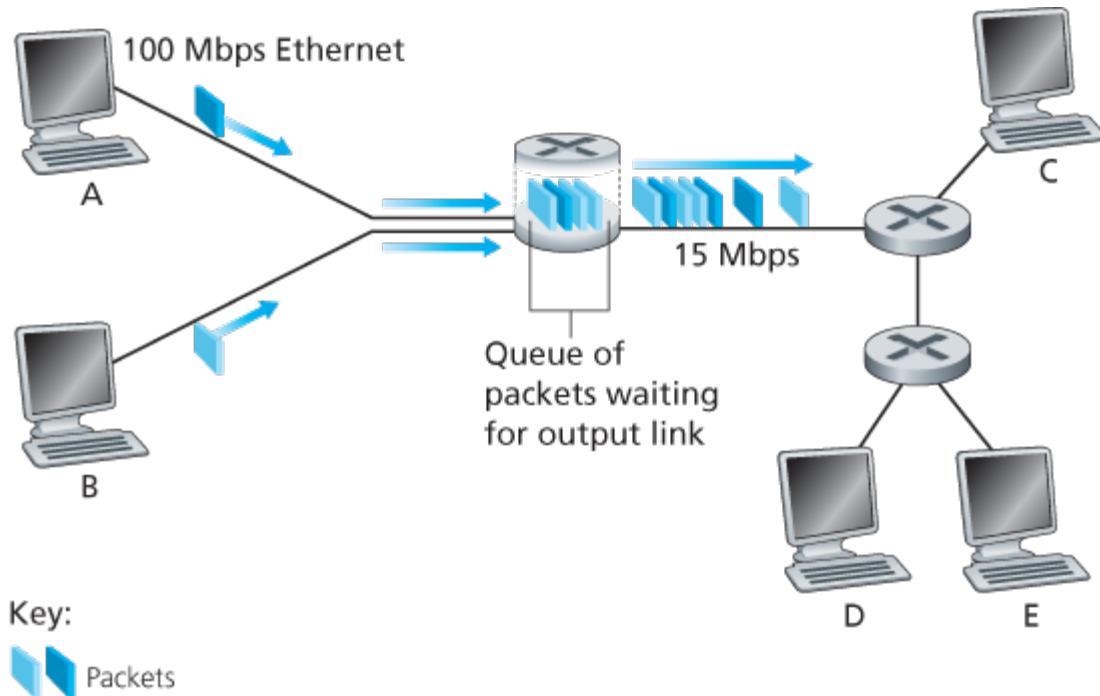


Figure 1.12 Packet switching

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 100 Mbps Ethernet links to the first router. The router then directs these packets to the 15 Mbps link. If, during a short interval of time, the arrival rate of packets to the router (when converted to bits per second) exceeds 15 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**.

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 5](#). 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 Trace-route 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.

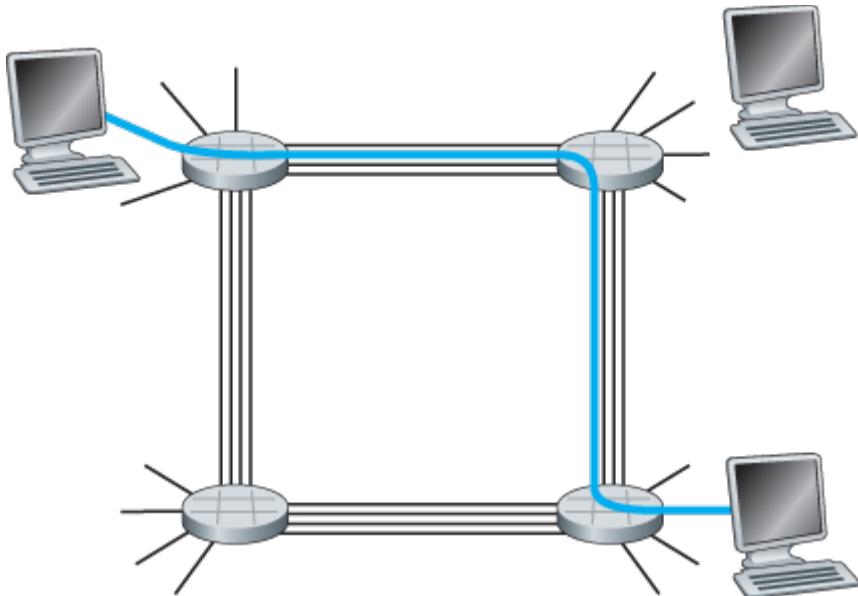


Figure 1.13 A simple circuit-switched network consisting of four switches and four links

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. 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.

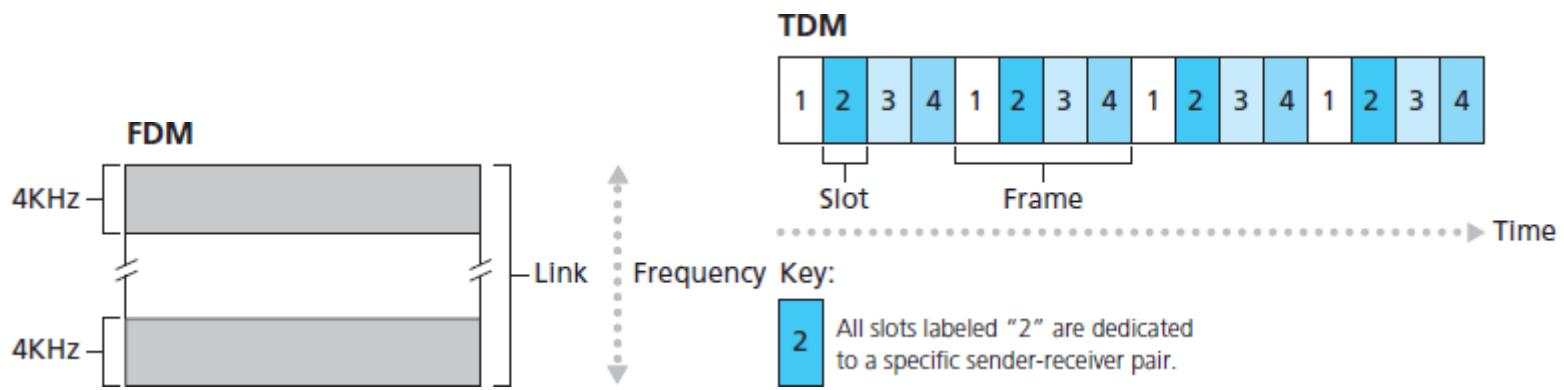


Figure 1.14

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)

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 each circuit is 64 kbps.

Proponents of packet switching have always argued that circuit switching is wasteful because the dedicated circuits are idle during **silent periods**. For example, when one person in a telephone call stops talking, the idle network resources (frequency bands or time slots in the links along the connection's route) cannot be used by other ongoing connections. As another example of how these resources can be 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)**, which is a meeting point where multiple ISPs can peer together. An IXP is typically in a stand-alone building with its own switches [\[Ager 2012\]](#). There are over 400 IXPs in the Internet today [\[IXP List 2016\]](#). 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 today’s Internet. 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 50–100 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 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 2.6](#).

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.

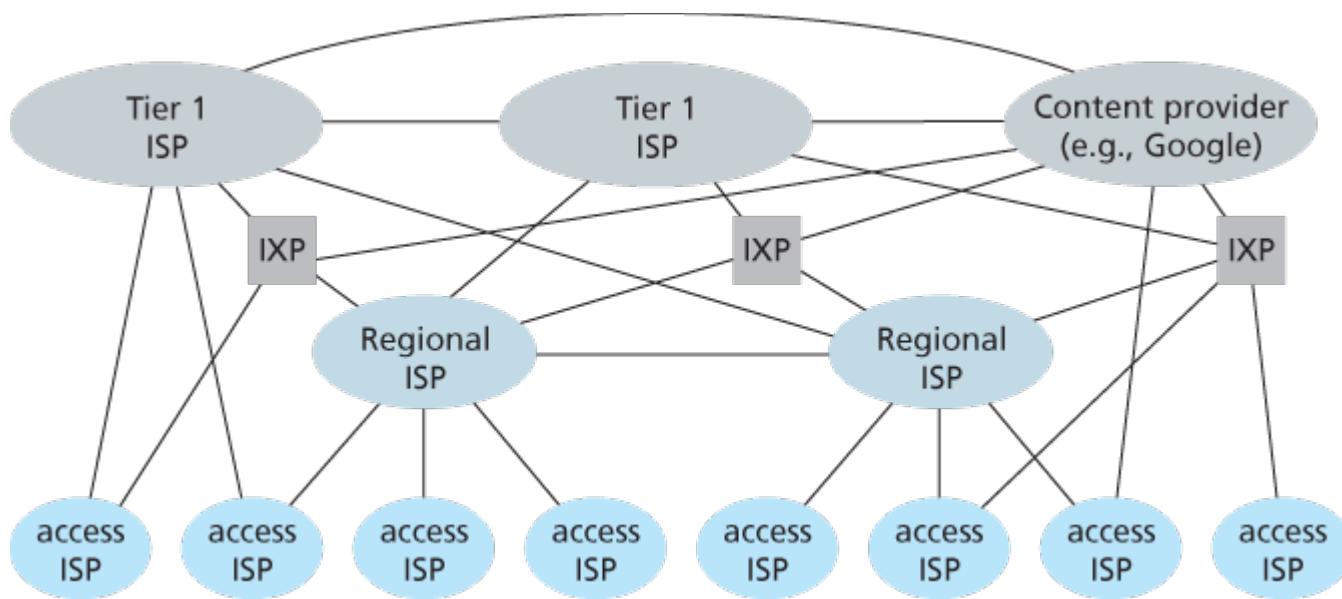


Figure 1.15 Interconnection of ISPs

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

Back in [Section 1.1](#) we said that the Internet can be viewed as an infrastructure that provides services to distributed applications running on end systems. Ideally, we would like Internet services to be able to move as much data as we want between any two end systems, instantaneously, without any 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 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, e-mail, 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

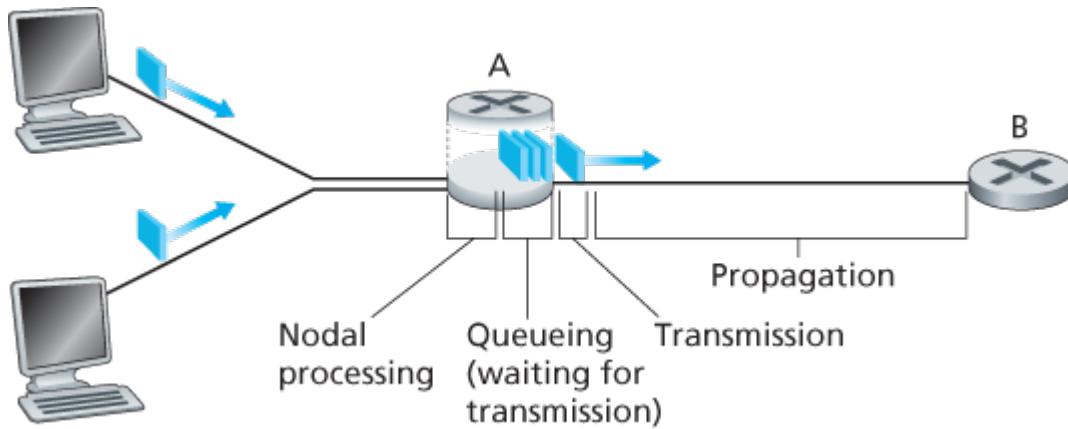


Figure 1.16 The nodal delay at router A

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

2·108 meters/sec to 3·108 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



Exploring propagation delay and transmission 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

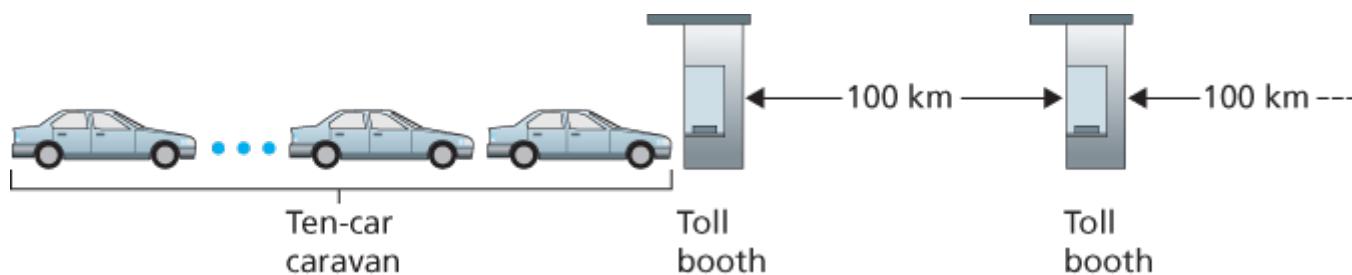


Figure 1.17 Caravan analogy

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

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

If a picture speaks a thousand words, then an animation must speak a million words. The 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](#), [Kleinrock 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](#).

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

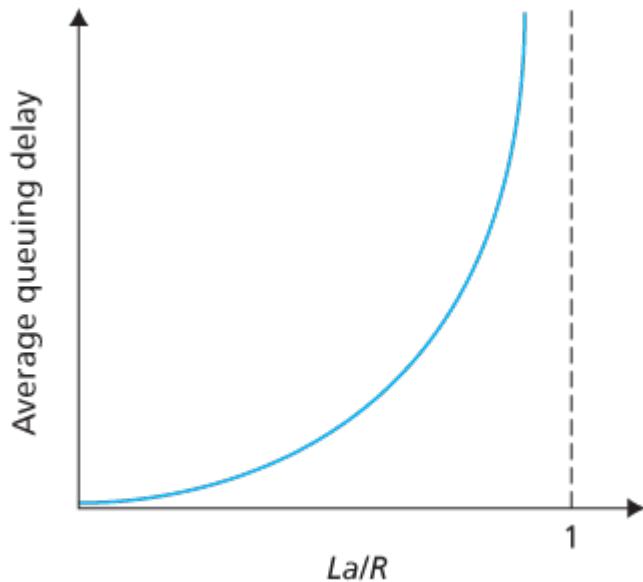


Figure 1.18 Dependence of average queuing delay on traffic intensity

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 textbook 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,

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

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



Using Traceroute to discover network paths and measure network delay

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  -acrl1-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* recommend 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 [\[PingPlotter 2016\]](#).

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 6](#). 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 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

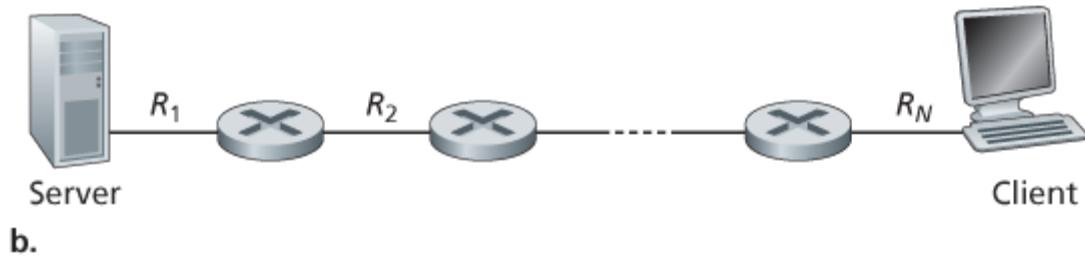
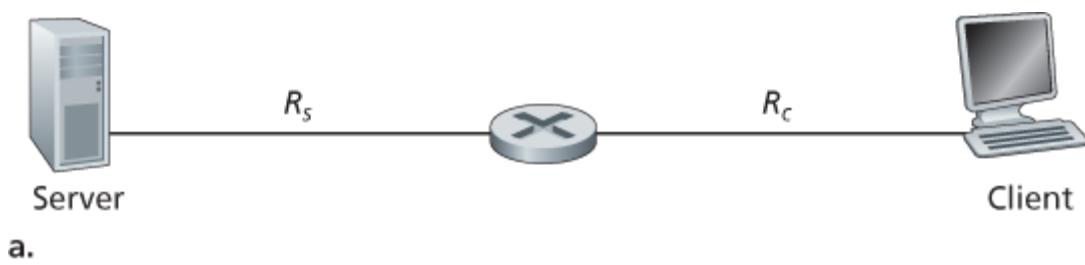


Figure 1.19 Throughput for a file transfer from server to client

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 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

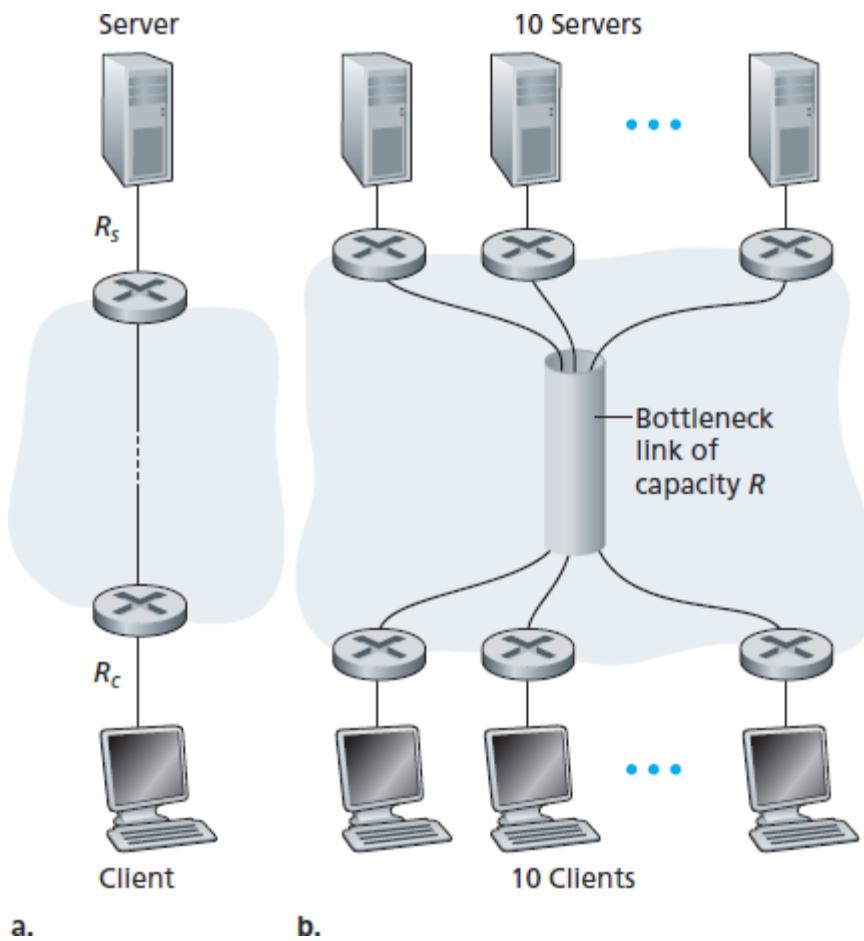


Figure 1.20 End-to-end throughput: (a) Client downloads a file from server; (b) 10 clients downloading with 10 servers

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**.

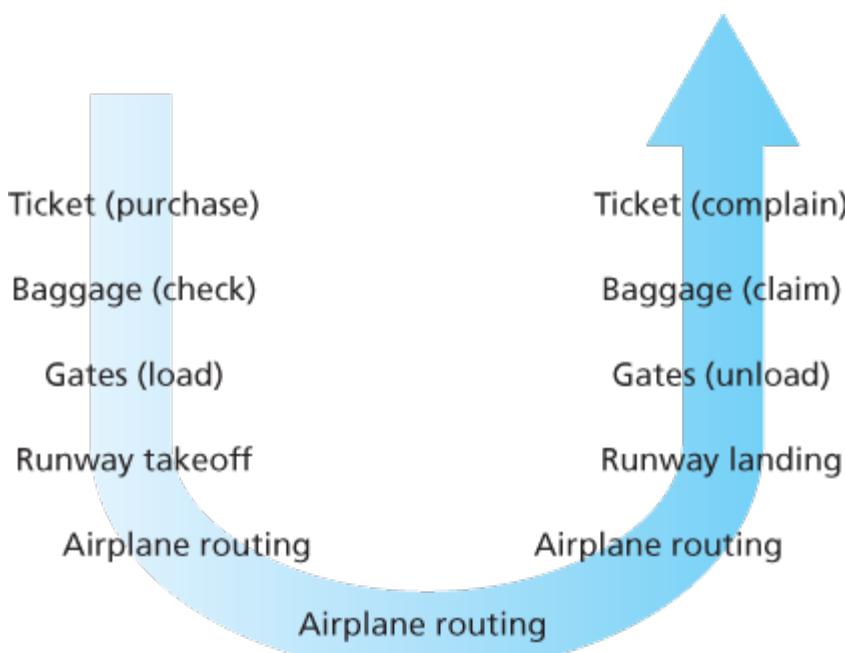


Figure 1.21 Taking an airplane trip: actions

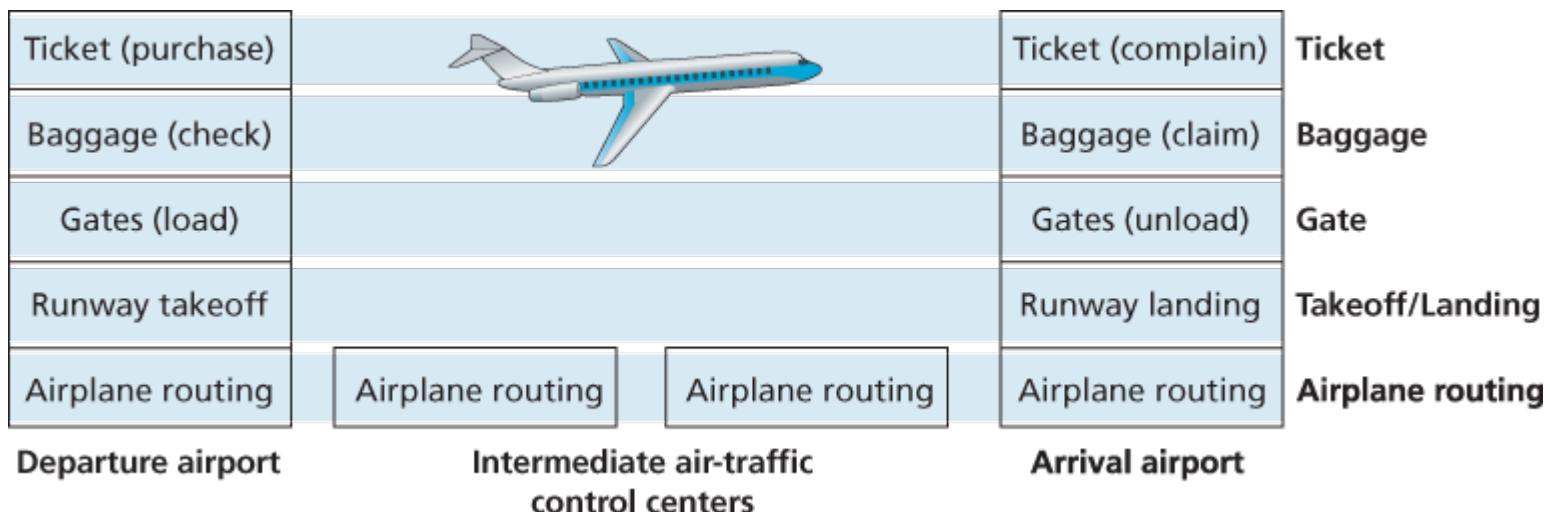


Figure 1.22 Horizontal layering of airline functionality

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 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 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

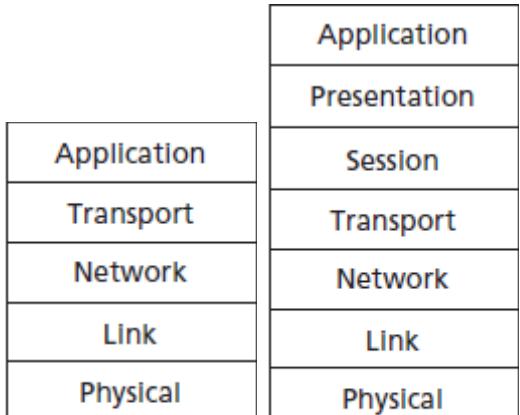
implementation of a service is very different from changing the service itself!) For example, if the gate functions were changed (for instance, to have people board and disembark by height), the remainder of the airline system would remain unchanged since the gate layer still provides the same function (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 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



a. Five-layer Internet protocol stack b. Seven-layer ISO OSI reference model

Figure 1.23 The Internet protocol stack (a) and OSI reference model (b)

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 2016]. 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 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

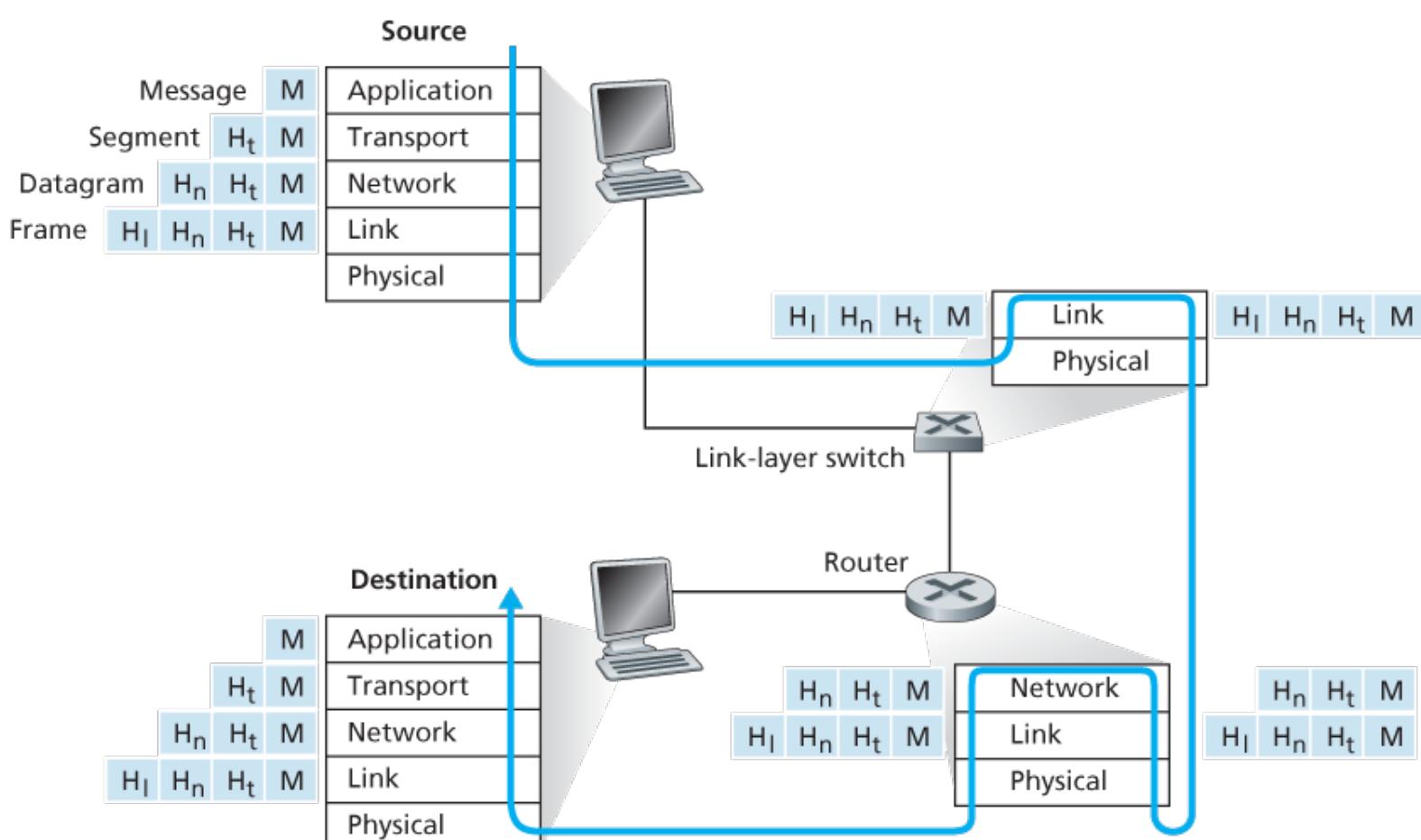


Figure 1.24 Hosts, routers, and link-layer switches; each contains a different set of layers, reflecting their differences in functionality

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 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. Billions of “things,” including wearables and home devices, are currently being connected to the Internet. 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 Instagram posts, Internet search results, streaming music, video conference calls, streaming movies, 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 and 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 [\[Moore 2001\]](#). The site Digital Attack Map allows use to visualize the top daily DoS attacks worldwide [\[DAM 2016\]](#). 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 service. DDoS attacks leveraging botnets with thousands of comprised hosts are a common occurrence today [DAM 2016]. 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.

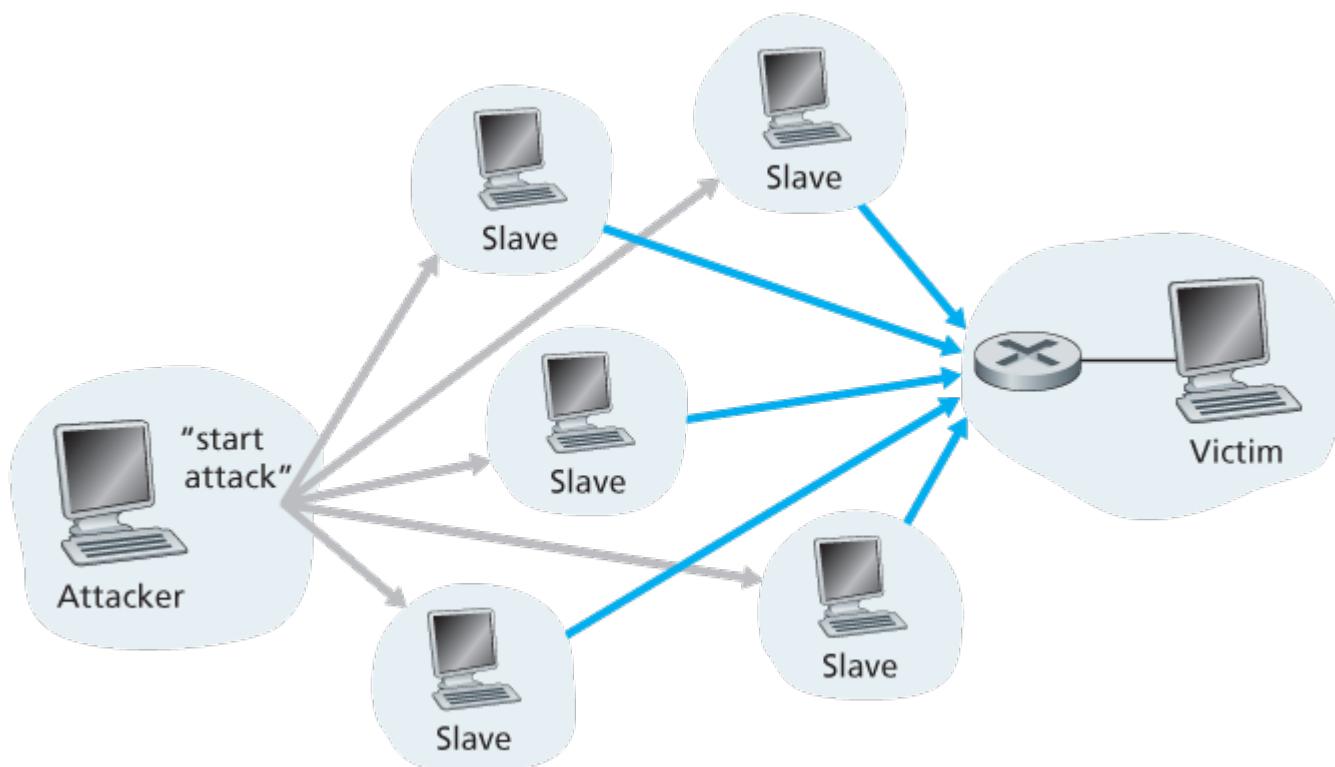


Figure 1.25 A distributed denial-of-service attack

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 7**). 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 [\[Wireshark 2016\]](#) 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 7](#)), 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, endpoint 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 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](#)]

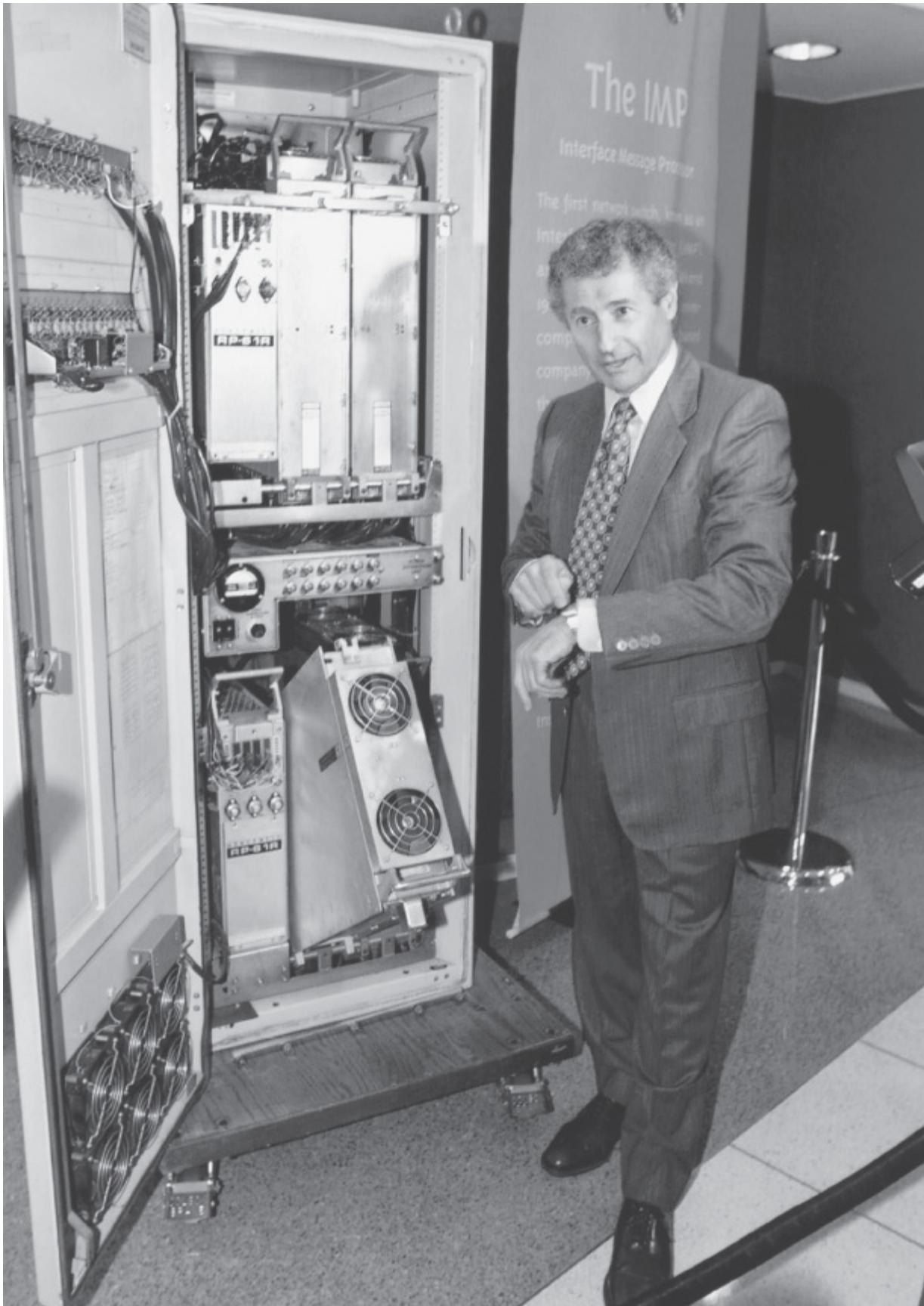


Figure 1.26 An early packet switch

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,

Facetime, and Google Hangouts).

- The increasing ubiquity of high-speed (54 Mbps and higher) public WiFi networks and medium-speed (tens of Mbps) Internet access via 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 such as Yelp, Tinder, Yik Yak, and Waz. 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, Instagram, Twitter, and WeChat (hugely popular in China)—have created massive people networks on top of the Internet. Many of these social networks are extensively used for messaging as well as photo sharing. Many Internet users today “live” primarily within one or more social networks. 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 e-mail 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., e-mail 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: Data Plane
5. Network Layer: Control Plane
6. The Link Layer and LANs
7. Wireless and Mobile Networks
8. Security in Computer Networks
9. Multimedia Networking

Chapters 2 through 6 are the five core chapters of this book. You should notice that these chapters are organized around the top four layers of the five-layer Internet protocol. 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 7 through 9**—zooms in on three enormously important (and somewhat independent) topics in modern computer networking. In **Chapter 7**, 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). **Chapter 8**, which addresses security in computer networks, first looks 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, which addresses multimedia networking, examines 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.

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 router, what information in the packet does the router 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:

- a. Would a packet-switched network or a circuit-switched network be more appropriate for this application? Why?
- b. 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.

- a. What is the maximum number of simultaneous connections that can be in progress at any one time in this network?
- b. Suppose that all connections are between switches A and C. What is the maximum number of simultaneous connections that can be in progress?
- c. 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.

- a. 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?
- b. 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.



VideoNote

Exploring propagation delay and transmission delay

- a. Express the propagation delay, d_{prop} , in terms of m and s .
- b. Determine the transmission time of the packet, d_{trans} , in terms of L and R .
- c. Ignoring processing and queuing delays, obtain an expression for the end-to-end delay.
- d. 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?
- e. Suppose d_{prop} is greater than d_{trans} . At time $t=d_{\text{trans}}$, where is the first bit of the packet?
- f. Suppose d_{prop} is less than d_{trans} . At time $t=d_{\text{trans}}$, where is the first bit of the packet?
- g. 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](#).)

- a. When circuit switching is used, how many users can be supported?
- b. For the remainder of this problem, suppose packet switching is used. Find the probability that a given user is transmitting.
- c. Suppose there are 120 users. Find the probability that at any given time, exactly n users are transmitting simultaneously. (*Hint:* Use the binomial distribution.)
- d. 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.

- a. What is N , the maximum number of users that can be supported simultaneously under circuit switching?
- b. 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 that 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 2.5·108m/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 λ denote traffic intensity; that is, $\lambda=La/R$. Suppose that the queuing delay takes the form $IL/R(1-I)$ for $I<1$.

- a. Provide a formula for the total delay, that is, the queuing delay plus the transmission delay.
- b. 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.



VideoNote

Using Traceroute to discover network paths and measure network delay

- 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?
- Repeat (a) but this time choose one city in France and another city in Germany.
 - 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 $R_{1k}, R_{2k}, \dots, R_{Nk}$. 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 over-night 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, $R \cdot d_{\text{prop}}$.

- b. 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?
- c. Provide an interpretation of the bandwidth-delay product.
- d. What is the width (in meters) of a bit in the link? Is it longer than a football field?
- e. 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.

- a. Calculate the bandwidth-delay product, $R \cdot d_{\text{prop}}$.
- b. 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?
- c. What is the width (in meters) of a bit in the link?

P28. Refer again to problem P25.

- a. How long does it take to send the file, assuming it is sent continuously?
- b. 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?
- c. 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.

- a. What is the propagation delay of the link?
- b. What is the bandwidth-delay product, $R \cdot d_{\text{prop}}$?
- c. 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 8·10⁶ 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.

- a. 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?
- b. 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?
- c. 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.

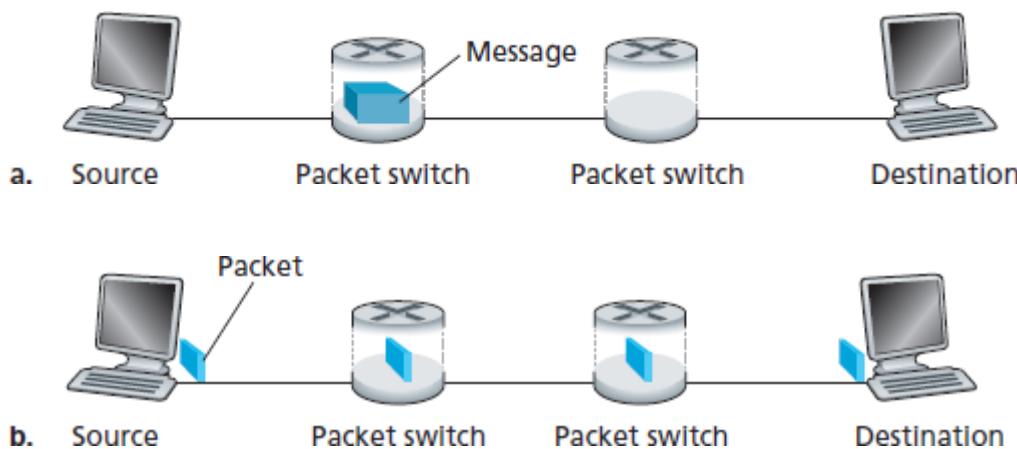


Figure 1.27 End-to-end message transport: (a) without message segmentation; (b) with message segmentation

- 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 computers.

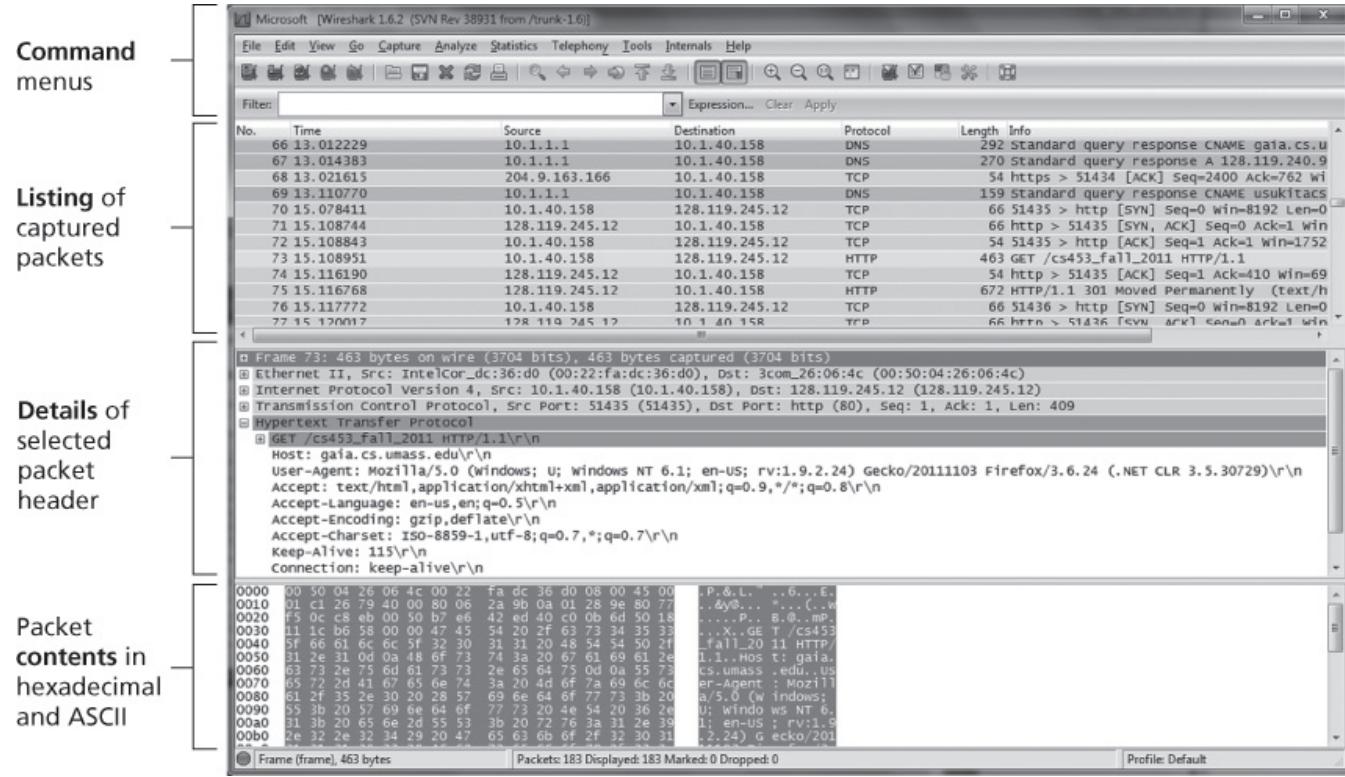


Figure 1.28 A Wireshark screenshot (Wireshark screenshot reprinted by permission of the Wireshark Foundation.)

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.pearsonhighered.com/cs-resources/>.

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 (e-mail, 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.

Chapter 2 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 infrastructure and protocols to support them. 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 e-mail, 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. In the new millennium, new and highly compelling applications continue to emerge, including voice over IP and video conferencing such as Skype, Facetime, and Google Hangouts; user generated video such as YouTube and movies on demand such as Netflix; multiplayer online games such as Second Life and World of Warcraft. During this same period, we have seen the emergence of a new generation of social networking applications—such as Facebook, Instagram, Twitter, and WeChat—which have created engaging human networks on top of the Internet's network of routers and communication links. And most recently, along with the arrival of the smartphone, there has been a profusion of location based mobile apps, including popular check-in, dating, and road-traffic forecasting apps (such as Yelp, Tinder, Waz, and Yik Yak). 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, peer-to-peer (P2P) file distribution, and video streaming.

([Chapter 9](#) will further examine multimedia applications, including streaming video and VoIP.) We then cover network application development, over both TCP and UDP. In particular, we study the socket interface 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.

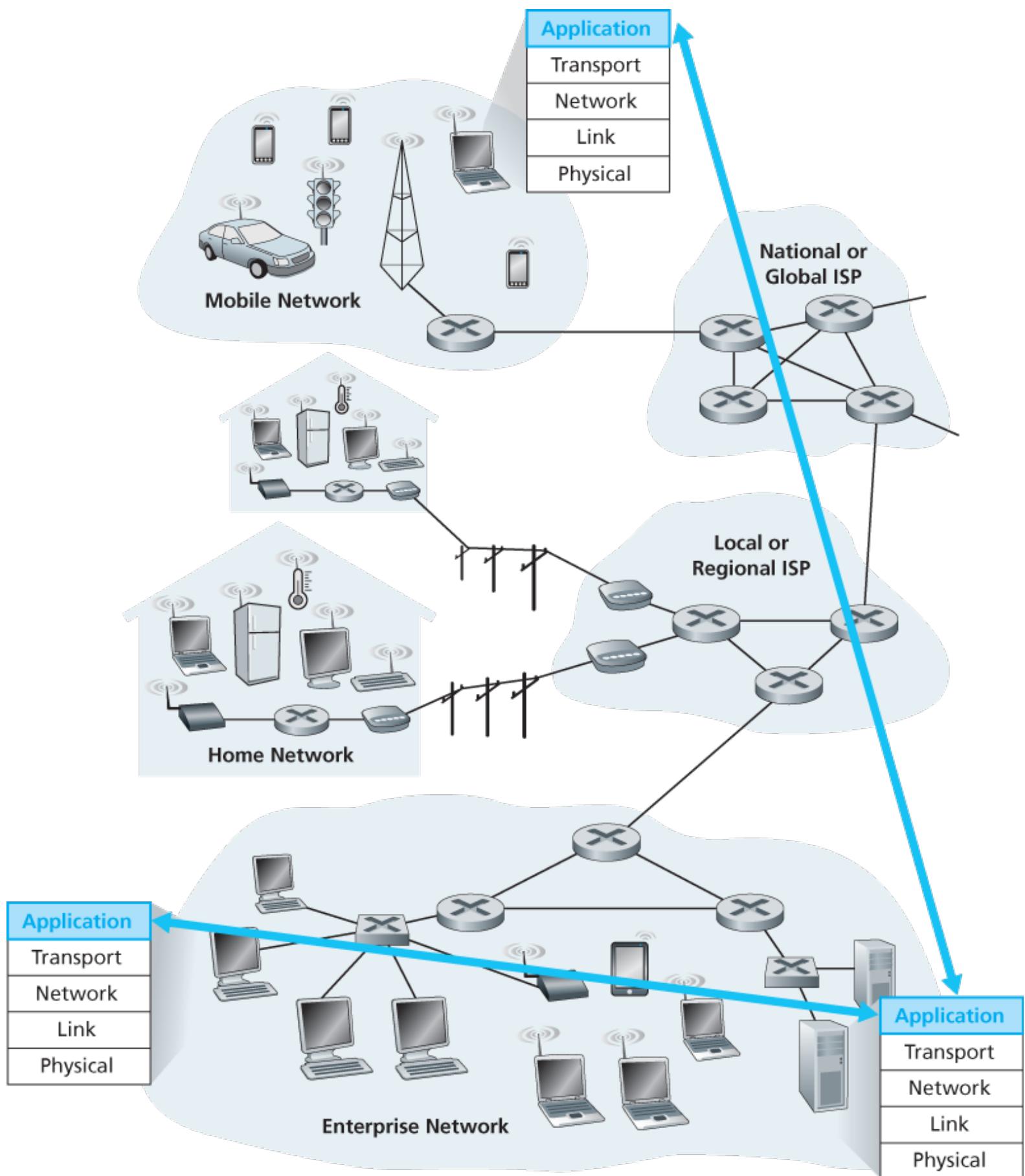


Figure 2.1 Communication for a network application takes place between end systems at the application layer

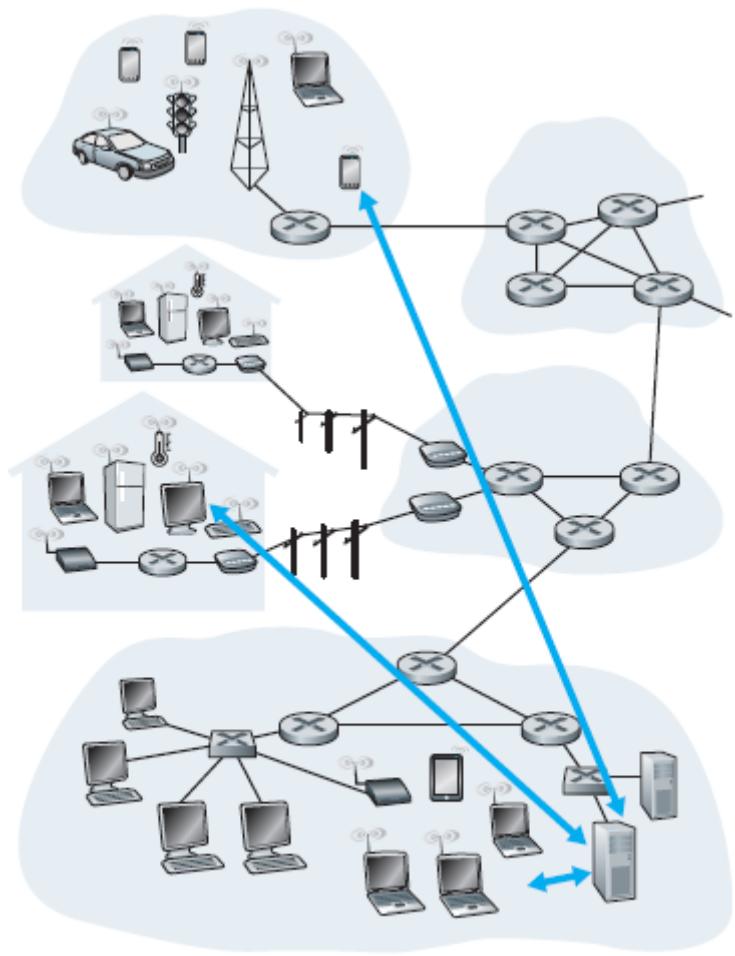
2.1.1 Network Application Architectures

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.

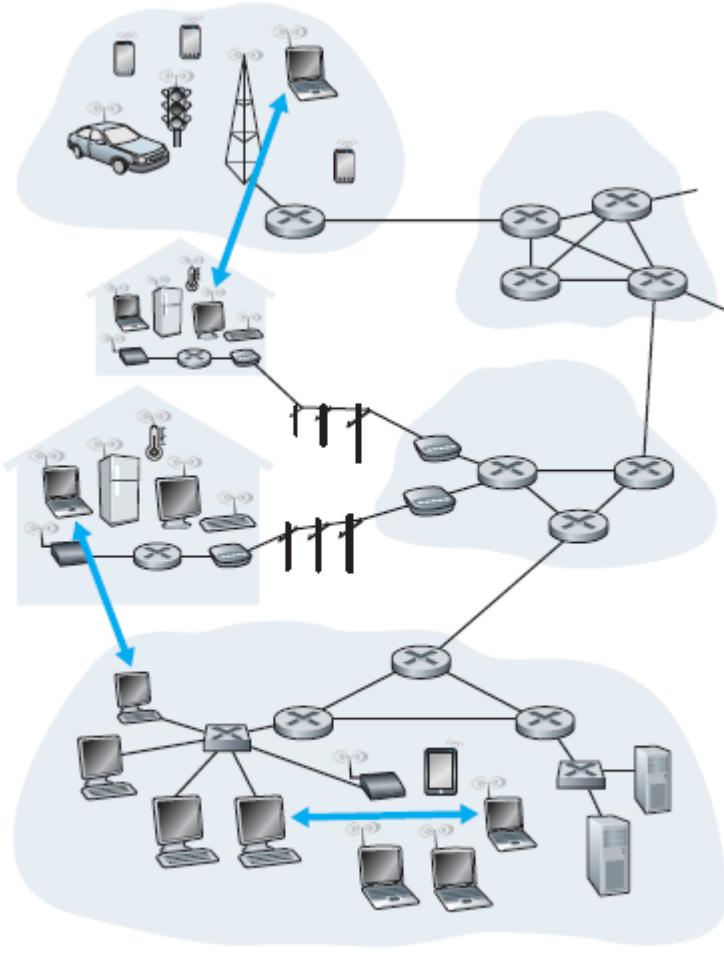
In a [client-server architecture](#), there is an always-on host, called the *server*, which services requests from many other hosts, called *clients*. A classic example is the Web application for which an always-on Web server services requests from browsers running on client hosts. When a Web server 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, Bing, Baidu), Internet commerce (e.g., Amazon, eBay, Alibaba), Web-based e-mail (e.g., Gmail and Yahoo Mail), social networking (e.g., Facebook, Instagram, Twitter, and WeChat)—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.

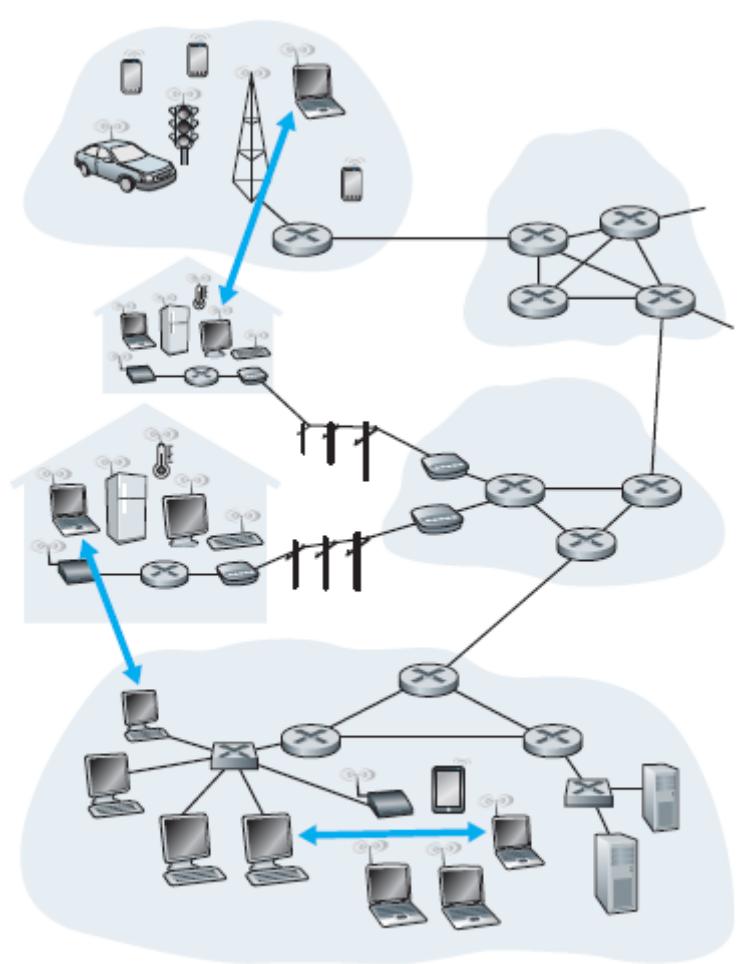
In a [P2P architecture](#), there is minimal (or no) reliance on dedicated servers in data centers. Instead the application exploits direct communication between pairs of intermittently connected hosts, called *peers*. The peers are not owned by the service provider, but are instead desktops and laptops controlled by users, with most of the



a. Client-server architecture



b. Peer-to-peer architecture



b. Peer-to-peer architecture

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 download acceleration (e.g., Xunlei), and Internet telephony and video conference (e.g., Skype). 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, P2P applications face challenges of security, performance, and reliability due to their highly decentralized structure.

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), 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.

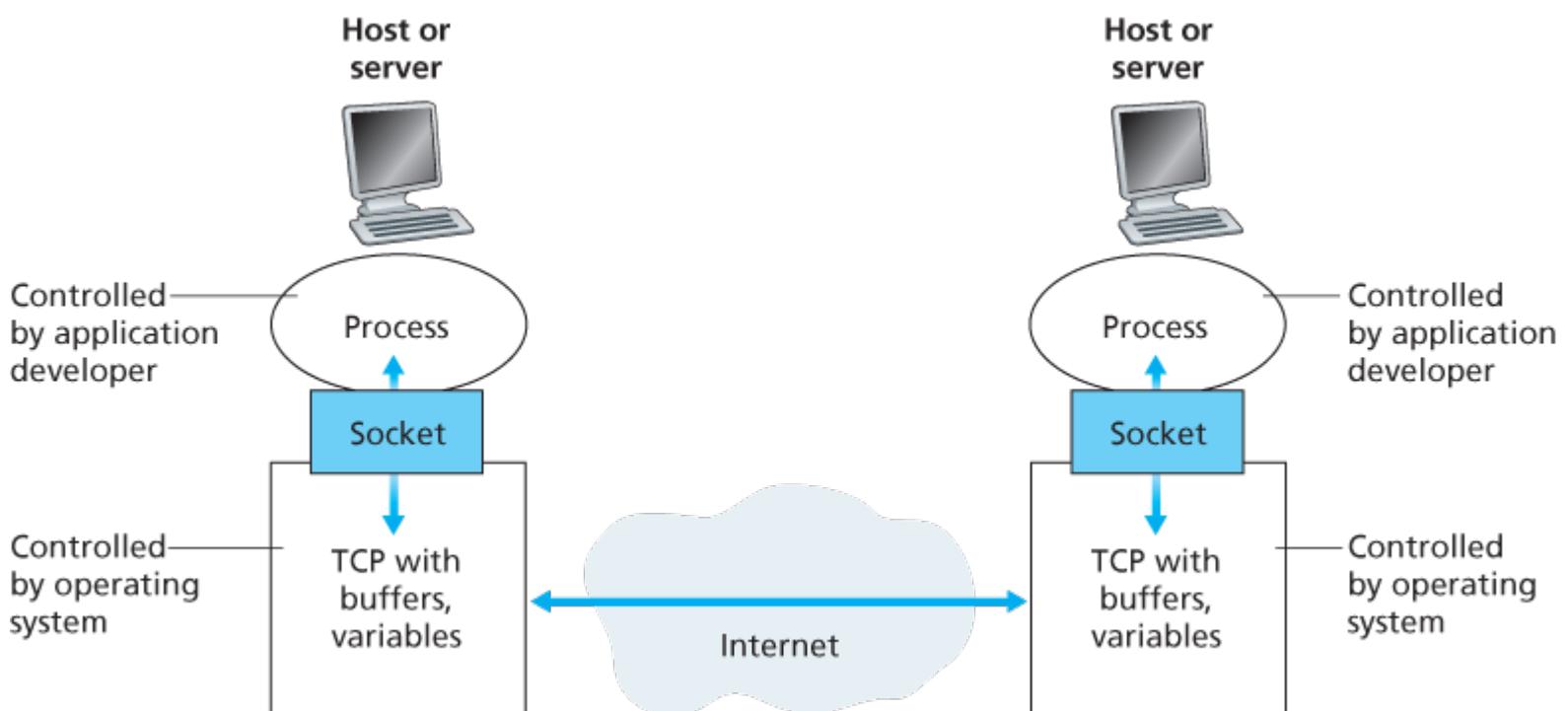


Figure 2.3 Application processes, sockets, and underlying transport protocol

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 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 9](#), [\[Gauthier 1999\]](#); [\[Ramjee 1994\]](#).) 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.

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

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
Smartphone messaging	No loss	Elastic	Yes and no

Figure 2.4 Requirements of selected network applications

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.

- **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

FOCUS ON SECURITY

SECURING TCP

Neither TCP nor UDP provides 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](#).

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