TABLE OF CONTENTS

PREFACE

OBJECTIVE

Communication networks are in a period of transition from networks that are based on telephone architecture and standards to networks based on the Internet Protocol (IP) architecture. The main reason for this change is that new services and applications can be deployed with unprecedented speed over an Internet that has attained the global reach of the telephone network. Many of these new applications and services are quite visible to the student. For example, in addition to e-mail and web surfing, there is buying/selling over the Internet (eBay), MP3 and other file exchanges (Napster, KaZaA), interactive games (Counterstrike), video streaming (CNN), and voice-over-IP (NetMeeting, ICQ). Many other applications and services are having profound impact on business, manufacturing, international commerce, medicine, and government.

The infrastructure of communication networks is undergoing dramatic change under pressure from the new services and enabled by technology innovation. A new generation of wireless devices combines aspects of cellular phones, personal digital assistants, and even digital cameras and is enabling new modes of mobile communication such as short text messaging, event notification, e-mail, and web browsing. These wireless services are blazing a trail away from traditional telephony to new modes of IP-based multimedia communications. Inevitably the signaling system that enables all of the functionality of the cellular and telephone network will be replaced by more versatile signaling based on Internet protocols. A new generation of IP-based protocols will control and manage the resources in the next generation of networks.

It is in this exciting context of new services and emerging next-generation network architecture that we offer this second edition of our textbook. The purpose of this book is to provide an introduction to fundamental network architecture concepts and their application in existing and emerging networks. We emphasize the pivotal role of Internet protocols in future network architecture and at the same time provide a broad coverage of fundamental network concepts. Our view is that the student will be best prepared for a future of constant change through exposure to network design alternatives.

TARGET COURSES

The book is designed for introductory courses in computer networks and in communication networks at the upper-level undergraduate and first-year graduate programs in electrical engineering, computer engineering, and computer science. The book contains all the basic material covered in typical one-semester first courses in computer networks and in communication networks. The book also provides additional material in each chapter as well as advanced topics in the later chapters so that the book can be used in a two-semester course sequence. The book is up-to-date in its coverage of emerging network architecture and can also be used by engineering and computer professionals.

As prerequisites the book assumes a general knowledge of computer systems. In certain optional parts of the text, knowledge of programming, elementary probability, or elementary calculus is required. These sections are clearly indicated and can be skipped.

APPROACH AND CONTENT

The book is organized into three parts. In Part I we trace the evolution of networks and identify the key concepts and functions that form the basis for layered architectures. We introduce examples of services and applications that are familiar to the student (web browsing, e-mail, and telephone), and we explain how these services are supported by networks. This *big picture view of networks* helps the student to see how the various parts of a network architecture fit into one whole.

Part I. The big picture of networks (Chapters 1–2):
- Evolution of network concepts in telegraph, telephone, and computer networks
- How services and applications drive network architecture evolution
- How layers work together to deliver services
- Application layer, transport layer, and sockets
- Preparation for experimentation: network protocol analyzer, IP utilities, and socket programming

The second part presents fundamental concepts of network architecture and focuses on the lower four layers of the protocol stack. Our approach is to develop the fundamental concepts first and then to show their application in concrete examples. For example, we develop ARQ in detail as an example of a peer-to-peer protocol and then we discuss its application in TCP reliable stream service and flow control. We cover the essential computer network topics, but we also have extensive discussion of access and transport networks, telephone and cellular services and signaling. This additional material is organized so that it is optional, but we include it because we believe that much of network architecture evolution in the near future will involve the extension of Internet protocols to traditional networks and services.

Part II. Fundamental concepts in network architecture (Chapters 3–7):
- Digital transmission; copper, cable, radio, and optical media
- SONET and optical networking
- Circuit switching, signaling, telephone services, mobility
- Peer-to-peer protocol design; PPP, HDLC, POS, and GFP
- Medium-access control; Ethernet and 802.11 wireless LANs
- Voice and data cellular networks
- Packet switching, routing, congestion control, and QoS

The third and final part deals with key network architectures, advanced topics, and next generation networks. We present the protocols and standards that are likely to shape the next generation networks.

Part III. Key architectures, advanced topics, and next generation networks (Chapters 8–12 and Appendices):

- IPv4 and IPv6; TCP, UDP; RIP, OSPF, BGP; DHCP and mobile IP
- ATM networks
- New Architectures: IntServ, RSVP, DiffServ, Peer vs. Overlay Interconnection, MPLS and GMPLS, RTP, SIP, and H.323
- Network Security: DES and AES, RSA, IPSec, SSL and TLS, 802.11
- Multimedia standards: JPEG, MPEG, Audio, MP3, voice-over-IP
- Network management and performance modeling

The book attempts to provide a *balanced view of all important elements of networking.* This is a very big challenge in the typical one-semester introductory course that has limited time available. We have organized the book so that all the relevant topics can be covered at some minimum essential level of detail. Additional material is provided that allows the instructor to cover certain topics in greater depth. Dependencies between sections are discussed later in the Preface.

CHANGES FROM THE FIRST EDITION

The most important change in the second edition is the extensive use of the open-source Ethereal network protocol analyzer in the teaching of network protocols. Ethereal allows any PC to capture live network traffic and to analyze the headers and payloads of the stream of captured traffic at layers 2 and above at any level of detail. We use Ethereal in the following ways:

- Examples of Ethereal packet captures demonstrate the operation of protocols such as HTTP, SMTP, Telnet, SIP, RSVP, RTP, DNS, TCP, UDP, OSPF, IP, ARP, Ethernet, 802.11, PPP, LCP, IPCP, SSL, and TLS.
- We provide instructors with Ethereal packet capture files that allow them to interactively demonstrate the operation of protocols in class.
- Exercises in the problem section require the students to carry out their own packet captures to examine and analyze the operation of protocols using real traffic.
- The book website contains experiments that involve the use of Ethereal in Linux-based router networks.

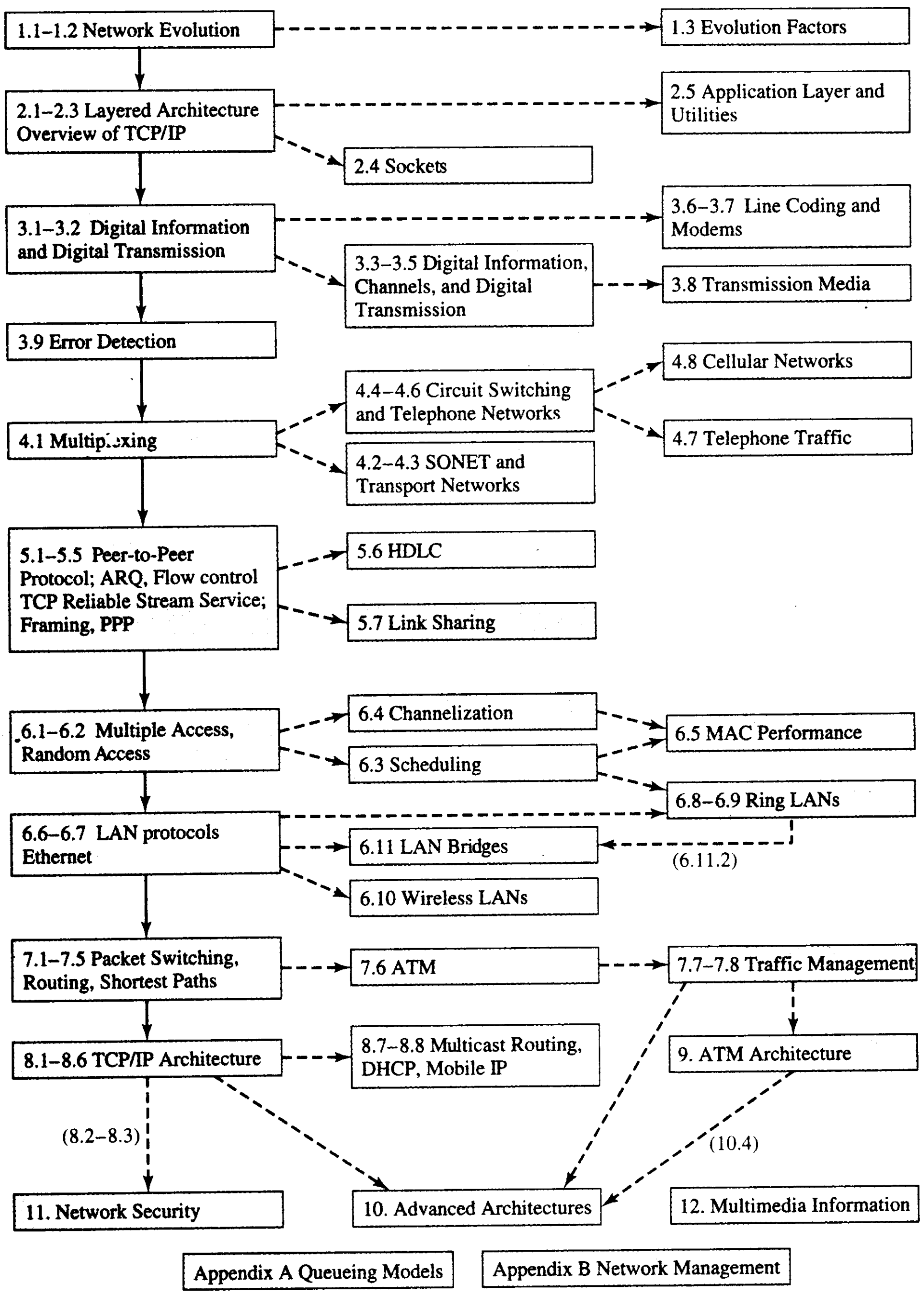
The second edition also contains various organizational and content changes. The following lists the key content and organizational changes from the first edition:

- The material in the book has been rearranged so that optional sections can be skipped without a disruption in the topic flow. The sections that contain optional material are indicated by a diamond (◆) in the heading. The optional sections that contain detailed mathematics are now indicated by a sidebar.
- Chapter 1 has been shortened and the discussion of network evolution has been simplified. The functions associated with each layer are introduced along with the discussion on network evolution.

- In Chapter 2 the discussion on how all the layers work together has been improved by introducing examples using Ethereal packet captures. The section on application layer protocols has been expanded and a new section provides an introduction to network protocol analyzers.
- PCM speech coding has been moved from Chapter 12 to Chapter 3.
- Chapter 4 provides more detail on SONET and optical transport networks. Satellite cellular networks have been dropped.
- Chapter 5 now consists of two parts. The first part deals with peer-to-peer protocols using reliable data transfer protocols as an example. The first part also includes TCP reliable byte stream service. The second part focuses on data link layer protocols and now includes a section on framing.
- Chapter 6 has also been divided into the principles of medium access control protocols (Part I) and LANs (Part II). We have simplified the mathematical discussion of medium access controls and provide details in a separate section.
- In Chapter 7 we have streamlined the discussion of packet networks, and we have separated clearly the more advanced discussion of traffic management.
- Chapter 8 makes extensive use of packet capture examples to illustrate the operation of TCP/IP protocols.
- Chapter 10 on advanced network architectures has been revised extensively. The discussion of ATM over IP has been replaced by a discussion of the overlay and peer models to network interconnection. The chapter now contains discussion on virtual networks and GMPLS. The material on RTP and SIP has been updated and moved from Chapter 12 to this chapter.
- Chapter 11 has been updated with brief discussions of the Advanced Encryption Standard and 802.11 security.

CHAPTER DEPENDENCIES

The book was designed to support a variety of introductory courses on computer and communication networks. By appropriate choice of sections, the instructor can provide a desired focus or make adjustments to account for the background of the students. Figure 1 contains a flow chart of the dependencies between sections in the book. The solid black line indicates the sequence of topics that forms the core of most introductory courses. The dashed lines show sections that can be added to provide greater depth in various topic areas. Section numbers in parentheses indicate dependencies that involve only subsections. After Chapter 8, the chapters are less interdependent. Chapter 11 depends on Chapter 8 only in regards to the header structure of IPv4 and IPv6. Chapter 12 depends only on Sections 3.1 and 3.2. Appendix A on queueing models provides supporting material for Sections 4.7, 5.7, and 6.5, which deal with performance modeling.



1.1–1.2 Network Evolution
1.3 Evolution Factors
2.1–2.3 Layered Architecture Overview of TCP/IP
2.5 Application Layer and Utilities
2.4 Sockets
3.1–3.2 Digital Information and Digital Transmission
3.6–3.7 Line Coding and Modems
3.3–3.5 Digital Information, Channels, and Digital Transmission
3.8 Transmission Media
3.9 Error Detection
4.8 Cellular Networks
4.4–4.6 Circuit Switching and Telephone Networks
4.1 Multiplexing
4.7 Telephone Traffic
4.2–4.3 SONET and Transport Networks
5.1–5.5 Peer-to-Peer Protocol; ARQ, Flow control TCP Reliable Stream Service; Framing, PPP
5.6 HDLC
5.7 Link Sharing
6.4 Channelization
6.1–6.2 Multiple Access, Random Access
6.5 MAC Performance
6.3 Scheduling
6.8–6.9 Ring LANs
6.6–6.7 LAN protocols Ethernet
6.11 LAN Bridges
(6.11.2)
6.10 Wireless LANs
7.1–7.5 Packet Switching, Routing, Shortest Paths
7.6 ATM
7.7–7.8 Traffic Management
8.1–8.6 TCP/IP Architecture
8.7–8.8 Multicast Routing, DHCP, Mobile IP
9. ATM Architecture
(8.2–8.3)
(10.4)
11. Network Security
10. Advanced Architectures
12. Multimedia Information
Appendix A Queueing Models
Appendix B Network Management

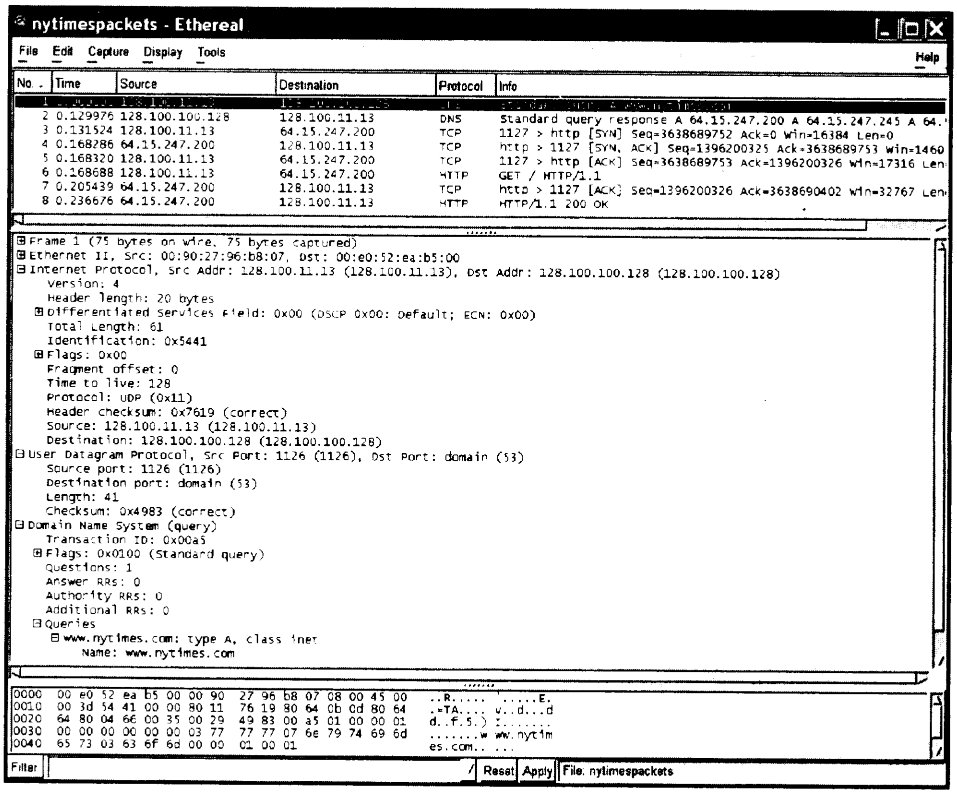
FIGURE 1 Chapter dependencies (solid arrows show sequence of core topics).

PEDAGOGICAL ELEMENTS

We have improved the pedagogical elements of the first edition and offer several new elements.

- *Ethereal Virtual Lab.* Lab experiments are the most effective means for reinforcing the concepts taught in lectures. At the physical layer a rich set of instruments (for example, oscilloscopes, spectrum analyzers, bit error rate sets) is available in the field to troubleshoot systems and can be used in the lab to teach and demonstrate the application of concepts in real systems. The tcpdump program written by Jacobson, Leres, and McCanne has formed the basis for many network protocol analyzers that are now used to troubleshoot protocols in network systems in the field and in the lab. The Ethereal open-source tool is one such network protocol analyzer that supports a very broad array of protocols and that is continuously updated by a large community of developers. We have used Ethereal to teach protocols in lectures and in experiments and have found it to be very effective in bringing the protocol concepts to life.

Figure 2 shows an Ethereal screenshot of a packet capture that a student can readily capture from a PC at home. The top pane shows the sequence of packets that transpire


nytimespackets - Ethereal
File Edit Capture Display Tools Help
Filter
Reset
Apply
File: nytimespackets

```
No. . Time     Source            Destination       Protocol  Info
  1 [illegible] 128.100.11.13   128.100.100.128   DNS       Standard query A www.nytimes.com
  2 0.129976 128.100.100.128   128.100.11.13     DNS       Standard query response A 64.15.247.200 A 64.15.247.245 A 64.
  3 0.131524 128.100.11.13     64.15.247.200     TCP       1127 > http [SYN] Seq=3638689752 Ack=0 Win=16384 Len=0
  4 0.168286 64.15.247.200     128.100.11.13     TCP       http > 1127 [SYN, ACK] Seq=1396200325 Ack=3638689753 Win=1460
  5 0.168320 128.100.11.13     64.15.247.200     TCP       1127 > http [ACK] Seq=3638689753 Ack=1396200326 Win=17316 Len
  6 0.168688 128.100.11.13     64.15.247.200     HTTP      GET / HTTP/1.1
  7 0.205439 64.15.247.200     128.100.11.13     TCP       http > 1127 [ACK] Seq=1396200326 Ack=3638690402 Win=32767 Len
  8 0.236676 64.15.247.200     128.100.11.13     HTTP      HTTP/1.1 200 OK
```

```
⊞ Frame 1 (75 bytes on wire, 75 bytes captured)
⊞ Ethernet II, Src: 00:90:27:96:b8:07, Dst: 00:e0:52:ea:b5:00
⊟ Internet Protocol, Src Addr: 128.100.11.13 (128.100.11.13), Dst Addr: 128.100.100.128 (128.100.100.128)
    Version: 4
    Header length: 20 bytes
  ⊞ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    Total Length: 61
    Identification: 0x5441
  ⊞ Flags: 0x00
    Fragment offset: 0
    Time to live: 128
    Protocol: UDP (0x11)
    Header checksum: 0x7619 (correct)
    Source: 128.100.11.13 (128.100.11.13)
    Destination: 128.100.100.128 (128.100.100.128)
⊟ User Datagram Protocol, Src Port: 1126 (1126), Dst Port: domain (53)
    Source port: 1126 (1126)
    Destination port: domain (53)
    Length: 41
    Checksum: 0x4983 (correct)
⊟ Domain Name System (query)
    Transaction ID: 0x00a5
  ⊞ Flags: 0x0100 (Standard query)
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  ⊟ Queries
    ⊟ www.nytimes.com: type A, class inet
        Name: www.nytimes.com
```

```
0000  00 e0 52 ea b5 00 00 90  27 96 b8 07 08 00 45 00   ..R..... '.....E.
0010  00 3d 54 41 00 00 80 11  76 19 80 64 0b 0d 80 64   .=TA.... v..d...d
0020  64 80 04 66 00 35 00 29  49 83 00 a5 01 00 00 01   d..f.5.) I.......
0030  00 00 00 00 00 00 03 77  77 77 07 6e 79 74 69 6d   .......w ww.nytim
0040  65 73 03 63 6f 6d 00 00  01 00 01                  es.com.. ...
```

FIGURE 2 Sample Ethereal display.

after the student clicks on a web link: A DNS query and response; A TCP three-way connection setup; an HTTP request, followed by a TCP acknowledgment, and then the beginning of the HTTP response. The sequence of packets exchanged for various types of protocols can be analyzed in this fashion. The middle pane allows the user to delve into the details in the headers of any of the protocol data units at layer 2 and above. The middle pane in the figure shows some of the details of the IP, UDP, and DNS headers. The bottom pane allows the user to zoom into the bits and bytes of the header as well as the payload. This rich set of capabilities was developed within Ethereal for use in troubleshooting protocols in development labs as well as in the field. These same capabilities make Ethereal an extremely powerful teaching tool.

We use Ethereal to examine PPP, HDLC, Ethernet, MPLS, IP, IPv6, OSPF, UDP, TCP, DNS, HTTP, RTP, SIP, H.323, SSL, and TLS. We provide the instructors with packet capture files that allow them to demonstrate protocols interactively in the classroom. Most of the examples can be readily reproduced and examined in greater detail by the student. We also introduce networks utilities such as PING, IPconfig, netstat, and traceroute, which can be used in exercises that involve Ethereal packet captures.

- *Numerous figures.* Network diagrams, time diagrams, performance graphs, state transition diagrams are essential to effectively convey concepts in networking.
- *Lecture charts.* We have prepared approximately 500 MS PowerPoint® charts for use in lecture presentations. We have also prepared approximately 50 presentation charts that use animation to demonstrate certain key concepts more effectively. All of these charts are available to instructors in the book web site.
- *Numerous examples.* The discussion of fundamental concepts is accompanied with examples illustrating the use of the concept in practice. Numerical examples are included in the text wherever possible.
- *Text boxes.* Commentaries in text boxes are used to discuss network trends and interesting developments, to speculate about future developments, and to motivate new topics.
- *Problems.* The authors firmly believe that learning must involve problem solving. The book contains approximately 600 problems. Each chapter includes problems with a range of difficulties from simple application of concepts to exploring, developing, or elaborating various concepts and issues. Quantitative problems range from simple calculations to brief case studies exploring various aspects of certain algorithms, techniques or networks. Programming exercises involving sockets and TCP/IP utilities are included where appropriate.
- *Chapter introductions.* Each chapter includes an introduction previewing the material covered in the chapter and in the context of the "big picture."
- *Chapter summaries and checklist of important terms.* Each chapter includes a summary that reiterates the most important concepts. A checklist of important terms aids the student in reviewing the material.
- *Mathematical sections.* In general key mathematical results are summarized in the main text. A sidebar indicates (optional) mathematical sections that contain more detailed mathematical discussion or derivation of these results.
- *References.* Each chapter includes a list of references. Given the introductory nature of the text, references concentrate on pointing to more advanced materials. Reference

to appropriate Internet Engineering Taskforce (IETF) RFCs and research papers is made where appropriate, especially with more recent topics.

- *A website.* The following website www.mhhe.com/leon-garcia contains links to the following teaching resources:
  - An *Instructor's Solutions Manual*
  - Additional problems, exercises and experiments for instructors
  - Answers to selected problems for students
  - Animated PowerPoint lectures and presentations
  - Chapter pointers to useful and interesting websites

ACKNOWLEDGMENTS

The material in the book was developed over many years in introductory as well as advanced courses in networking, both in regular undergraduate and graduate programs as well as in programs with an orientation towards professional practice. We acknowledge the feedback from the many students who participated in these courses and who used various versions of the manuscript.

For their help on *the first edition*, we acknowledge the input of the graduate students who served as teaching assistants in these courses, especially Dennis Chan, Yasser Rasheed, Mohamed Arad, Massoud Hashemi, Hasan Naser, and Andrew Jun.

We thank Anindo Banerjea, Raouf Boutaba, Michael Kaplan and Gillian Woodruff for many exciting conversations on networking. Anindo and Raouf graciously provided some of the material that is presented in Chapter 2. We would also like to thank Anwar Elwalid and Debasis Mitra for their continued encouragement and interest in the book. We thank Yau-Ren Jenq for reviewing the fair queueing discussions in detail.

We are especially grateful to Irene Katzela for testing the manuscript in her courses. We also thank Ray Pickholtz for testing various versions of the text, including the beta version, and for his many valuable suggestions and his continued encouragement.

We thank the reviewers for their many useful comments on the various versions of the first edition manuscript: Subrata Banerjee (Stevens Institute of Technology), John A. Copeland, (Georgia Institute of Technology), Mario Gerla (UCLA), Rohit Goyal (Ohio State University), Gary Harkin (Montana State University), Melody Moh (San Jose State University), Kihong Park (Purdue University—West Lafayette), Raymond L. Pickholtz (The George Washington University), Chunming Qiao (SUNY Buffalo), Arunabha Sen (Arizona State University), Stuart Tewksbury (West Virginia University), and Zhi-li Zhang (University of Minnesota).

We would like to acknowledge the many friends from Nortel Networks for showing us the many facets of networking. We thank Sidney Yip for opening the door to many years of interaction. We also thank Richard Vickers, Marek Wernik, and Jim Yan for many illuminating conversations over the years. We especially thank Tony Yuen for sharing his vast knowledge of the networking industry and for continuously showing how the big picture is actually bigger!

For their help with the preparation of *the second edition* we would like to thank the following reviewers: Maurice F. Aburdene (Bucknell University), John Anderson

(University of Lund), LeRoy W. Bearnson (Brigham Young University), Michael J. Donahoo (Baylor University), Yuguang Fang (University of Florida), Jeff Fedor (University of Southern California), Jeremiah F. Hayes (Concordia University), Mike Kain (Drexel University/Unisys Corporation), W. David Laverell (Calvin College), Ching-Cheng Lee (San Jose State University), Derong Liu (University of Illinois—Chicago), T. N. Nagabhushan (Sri Jayacharajendra College of Engineering), Juan Pimentel (Kettering University), Arunabha Sen (Arizona State University), Vassilis Tsaoussidis (Northeastern University), and Bryan Usevitch (Univ. of Texas at El Paso).

We also thank many friends at AcceLight Networks who shared their experience of networks with us: Hyong Kim, Leigh Chang, Louis Zhang, Massoud Hashemi, Paul Chow, Debashis Basak, Denny Lee, Safa Almalki, Tom McGill, Hasan Naser, Srini Ramaswami, Stanko Vuleta, Rick Murphy, Stephane Gagnon, Ryan Moore, Mark Milinkovic, Jingyun Zhang, and Mark Fredericks. We thank Paul Bertels and Muhamed Durani for sharing their knowledge of Ethereal with us. We are also grateful for various technical interactions we had with colleagues at Fujitsu: Yau-Ren Jenq, Edmund Cher, Amalendu Chatterjee, Anna Cui, Steven Wright, and Masafumi Katoh; and at Bell Labs: Iraj Saniee, Yikai Su, Debasis Mitra, Anwar Elwalid, Carl Nuzman, Krishnan Kumaran, Randy Giles, and Lijun Qian.

We thank Eric Munson from McGraw-Hill for persuading us to take the plunge with this project, and Betsy Jones, Publisher, for providing decisive support at key times. In addition we thank the production team at McGraw-Hill for their patience, ideas, and continued support, especially John Griffin and Jill Peter.

IW would like to thank his wife Liesye for her constant encouragement and continued support while he was working on the second edition of the book.

Finally, ALG would like to thank his partner and best friend, Karen Carlyle, who made this second edition happen. ALG can finally think about renovating the house.

With the help of the many reviewers, professors, and students who have used previous versions of this book, we have tried to make the complex and fluid topic of network architecture as approachable, up-to-date, and error-free as possible. We welcome all comments and suggestions on how to improve the text. Please contact us via the text's website with any ideas you may have.

Alberto Leon-Garcia
Indra Widjaja

ABOUT THE AUTHORS

Alberto Leon-Garcia is a Professor in the Department of Electrical and Computer Engineering at the University of Toronto where he holds the Jeffrey Skoll Chair in Computer Networks and Innovation. He was also Chief Technical Officer and co-founder of AcceLight Networks Inc. where he led the development of a terabit multi-service optical switch. In 1999 Dr. Leon-Garcia became an IEEE fellow for "*For contributions to multiplexing and switching of integrated services traffic*."

At the University of Toronto, Dr. Leon-Garcia was the first holder of the Nortel Institute Chair in Network Architecture and Services. In 1998 he founded the Master of Engineering in Telecommunications program. Dr. Leon-Garcia has proudly supervised more than sixty graduate and postgraduate students.

Dr. Leon-Garcia is the author of the textbooks *Probability and Random Processes for Electrical Engineering* (Addison-Wesley), and *Communication Networks: Fundamental Concepts and Key Architectures* (McGraw-Hill), coauthored with Dr. Indra Widjaja.

Indra Widjaja received his Ph.D. degree in electrical engineering on high-speed packet switching architectures from the University of Toronto in 1992. Since 2001, he has been a researcher at Bell Laboratories, Lucent Technologies.

Dr. Widjaja's current research interests include traffic engineering, architectures for cost-effective transport networks, and high-speed packet switching. He has extensive publication in technical journals and conferences and holds several patents in switching architectures. He is also an active member of IETF and IEEE.

In 1993, Dr. Widjaja performed research on traffic management at the Teletraffic Research Center in Adelaide, Australia. From 1994 to 1997, he was Assistant Professor of Electrical and Computer Engineering at the University of Arizona where he taught courses in Computer Networking, Computer Architecture, and Digital Telephony, and conducted research in communication networks. He was also a Technical Consultant to Motorola and Lucent Technologies. From 1997 to 2001, he was with Fujitsu Network Communications where he worked on the architecture definitions and requirements for core switch, access switch, and transport products.

CHAPTER 1

Communication Networks and Services

A communication network, in its simplest form, is a set of equipment and facilities that provides a **service:** the transfer of information between users located at various geographical points. The most familiar example of a communication network is the telephone network, which provides telephone service, the bidirectional transfer of voice signals between people. Other examples of networks include computer networks, television broadcast networks, cellular networks, and the Internet.

Communication networks provide a service much like other ubiquitous utilities such as the water supply or electricity power systems. On the other hand, communication networks exhibit tremendous flexibility in their use and in this respect they are closest to transportation networks. Communication networks, along with transportation networks, have become essential infrastructure in every society. Both types of networks provide flexible interconnectivity that allows the flow of people and goods in the case of transportation and information in the case of communications. Both transportation and communication networks are "enabling" in that they allow the development of a multiplicity of new services. For example, the development of a postal service presupposes the availability of a good transportation system. Similarly, the availability of telephone service enables other services such as facsimile, voice mail, and electronic banking.

The ability of modern communication networks to transfer communication at extremely high speeds allows users to *gather information* in large volumes nearly instantaneously and, with the aid of computers, to almost immediately exercise *action at a distance*. These two unique capabilities form the basis for many emerging services and an unlimited number of future network-based services. For example, the Internet currently provides the communication services that enable computers and servers to provide valuable services such as e-mail, instant messaging, information search and retrieval, and various forms of electronic commerce. Audio streaming and interactive network games are indicators of the enormous potential for media rich services over a future Internet.

The purpose of this chapter is to explain how the design or "architecture" of a network is influenced by the services that it supports. In Section 1.1 we consider the evolution of three example networks and their associated services. This discussion serves to identify the basic elements that networks must provide. In Section 1.2 we return to a discussion of new and emerging services and we consider how these services are influencing the evolution of modern networks. Section 1.3 concludes the chapter by examining other factors that influence network evolution.

1.1 EVOLUTION OF NETWORK ARCHITECTURE AND SERVICES

A communication service involves the transfer of information. Different services differ in the details of how and in what form information is transferred. We will use three example networks to show how the details of the service influence the design of the network. The three example networks are: telegraph networks, telephone networks, and computer networks. In each case we provide an overview of the network's development, and then discuss the network from a more general service/architecture viewpoint. An indicator of the progress in communications technology is the speed at which information can be transmitted as measured in bits/second. Figure 1.1 shows the improvement in transmission bit rate over the last 150 years. In this time we have gone from telegraph systems that operated at tens of bits/second to modern optical systems that operate at terabits/second.

1.1.1 Telegraph Networks and Message Switching

In 1837 Samuel B. Morse demonstrated a practical telegraph that provided the basis for *telegram service,* the transmission of text messages over long distances. In the Morse telegraph, shown in Table 1.1, the text was encoded into sequences of dots and dashes. Each dot or dash was communicated by transmitting short and long pulses of electrical current over a copper wire. By relying on two signals, telegraphy made use of a **digital**

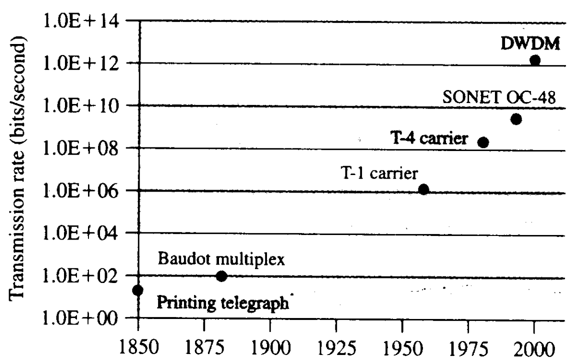


FIGURE 1.1 Evolution of transmission rate.

TABLE 1.1 International Morse code.

	Morse Code		Morse Code		Morse Code		Morse Code
A	·—	J	·———	S	···	2	··———
B	—···	K	—·—	T	—	3	···——
C	—·—·	L	·—··	U	··—	4	····—
D	—··	M	——	V	···—	5	·····
E	·	N	—·	W	·——	6	—····
F	··—·	O	———	X	—··—	7	——···
G	——·	P	·——·	Y	—·——	8	———··
H	····	Q	——·—	Z	——··	9	————·
I	··	R	·—·	1	·————	0	—————

transmission system. The Morse telegraph system is a precursor of the modern digital communication system in which all transmission takes place in terms of binary signals and all user information must first be converted to binary form.

In 1851 the first submarine cable was established between London and Paris. Eventually, *networks of telegraph stations* were established, covering entire continents. In these networks a message or telegram would arrive at a telegraph station, and an operator would make a **routing** decision based on the destination **address.** The operator would *store* the message until the desired communication line became available and then would *forward* the message to the next appropriate station. This **store-and-forward** process would be repeated at each intermediate station until the message arrived at the destination station. **Message switching** is used to describe this approach to operating a network. Addressing, routing, and forwarding are essential elements of modern computer networks.

The **transmission rate** (in bits/second) at which information could be transmitted over a telegraph circuit was initially limited to the rate at which a single human operator could enter a sequence of symbols. An experienced operator could transmit at a speed of 25 to 30 words/minute, which, assuming five characters per word and 8 bits per character, corresponds today to 20 bits/second (bps) in Figure 1.1.

A subsequent series of inventions attempted to increase the rate at which information could be transmitted over a single telegraph circuit by **multiplexing,** that is, combining the symbols from several operators onto the same communication line. One multiplexing system, the Baudot system invented in 1874, used *characters,* groups of five binary symbols, to represent each letter in the alphabet. The Baudot multiplexing system could interleave characters from several telegraph operators into a single transmission line.

The Baudot system eventually led to the modern practice of representing alphanumeric characters by groups of binary digits as in the ASCII code (short for American Standard Code for Information Interchange). Table 1.2 shows the binary representation for a subset of the ASCII code. The Baudot system also eventually led to the development of the teletype terminal, which provided a keyboard for entering character information, and could be used to transmit and receive digital information. The terminal was later used as one of the early input/output devices for digital computer systems. As Figure 1.1 shows, a Baudot multiplexer telegraph with six operators achieved a speed of 120 bps.

TABLE 1.2 Subset of ASCII character set and their binary representation.

Binary	Char	Binary	Char	Binary	Char
0000000	NUL	0110000	0	1000001	A
0000001	SOH	0110001	1	1000010	B
0000010	STX	0110010	2	1000011	C
0000011	ETX	0110011	3	1000100	D
0000100	EOT	0110100	4	1000101	E
0000101	ENQ	0110101	5	1000110	F
0000110	ACK	0110110	6	1000111	G
0000111	BEL	0110111	7	1001000	H
0001000	BS	0111000	8	1001001	I
0001001	HT	0111001	9	1001010	J
...		...		...	

Another approach to multiplexing involves *modulation*, which uses sinusoidal signals (tones) to carry multiple telegraphy signals. For example, each of the binary symbols could be transmitted by sending a sinusoidal signal of a given frequency for a given period of time, say, frequency f_0 to transmit a "0", f_1 to transmit a "1." Multiple sequences of binary symbols could be transmitted simultaneously by using multiple pairs of frequencies for the various telegraphy signals. These modulation techniques formed the basis for today's modems.

Prior to the invention of telegraphy, long-distance communication depended primarily on messengers that traveled by foot, horse, or other means. In such systems, a message might propagate at a rate of tens of kilometers per day. The invention of the electric telegraph made long-distance communication almost instantaneous; for example, an electric signal would take less than 5 milliseconds to cover 1000 kilometers.[1] Clearly, electrical communications had marked advantages over all other forms of communications. Indeed, the telegraph gave birth to the "news" industry; to this day, some newspapers have the name "The Daily Telegraph." In 1901 Guglielmo Marconi used Morse code to send and receive radio signals across the Atlantic, giving birth to long-distance wireless radio communications.

Now let us consider the telegraph network from a service/architecture viewpoint. The telegraph service involves the transmission of text messages between geographically distant locations. To provide this service, the architecture of the telegraph network contains the following key elements or functions:

1. The foundation for this network is a *digital transmission* system that enables two digits to be sent, "dot" and "dash" in the case of Morse code, or "zero" and "one" in the case of Baudot or ASCII code. The transmission medium can be copper wire, radio, or smoke signals for that matter.[2]

[1]The speed of light in a vacuum is 3×10^8 meters/second; in cable, it is 2.3×10^8 meters/second; in optical fiber, it is 2×10^8 meters/second.

[2]Indeed in the 1700s "visual telegraph" networks in Europe used line-of-sight semaphore systems to send signals between towers and hills. Visual networks of this type date back at least to the time of the Roman Empire.

2. A *framing* method is required for indicating the beginning and end of messages and for taking the sequence of dots/dashes or zeros/ones and grouping them into characters, and in turn, meaningful messages.
3. A system for specifying the destination *address* of messages is needed. A *routing* procedure determines the path that a message follows across a network of telegraph stations interconnected by digital transmission lines.

We will see later that the above elements and functions are fundamental to the design of modern computer networks.

1.1.2 Telephone Networks and Circuit Switching

In 1875, while working on the use of sinusoidal signals for multiplexing in telegraphy, Alexander Graham Bell recognized that direct transmission of a voice signal over wires was possible. In 1876 Bell developed a device that could transmit the entire voice signal and could form the basis for voice communication, which we now know as the *telephone*. The modern telephone network was developed to provide basic **telephone service,** which involves the two-way, real-time transmission of voice signals across a network.

The telephone and telegraph provided services that were fundamentally different: The telegraph required an expert operator with knowledge of Morse code, while the telephone terminal was very simple and did not require any expertise. Consequently the telephone was targeted as a direct service to end users, first in the business and later in residential markets. The deployment of telephones grew quickly, from 1000 phones in 1877 to 50,000 in 1880 and 250,000 in 1890.

Connectivity in the original telephone system was provided by an **analog transmission system.** The transmitted electrical signal is analogous to the original voice signal, that is, the signal is proportional to the sound pressure in speech as shown in Figure 1.2. The task of the analog transmission system is to deliver a signal that is a replica of the original voice signal. The need to preserve the quality of the signal led to the development of methods for conditioning transmission lines and for amplifying signals so that longer reaches could be attained.

It was quickly recognized in the early days of telephony that providing dedicated lines between each pair of users is very costly. As shown in Figure 1.3a, $N(N-1)/2$ transmission lines are required if dedicated lines are deployed between each pair of N users. The number of transmission lines grows very quickly, for example $N = 1000$ requires 495,000 lines! In 1878 **telephone switches** were introduced to allow human operators to interconnect telephone users on demand.

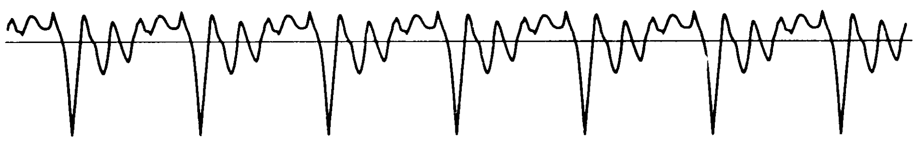


FIGURE 1.2 Sample signal of the sound "ae" as in cat.

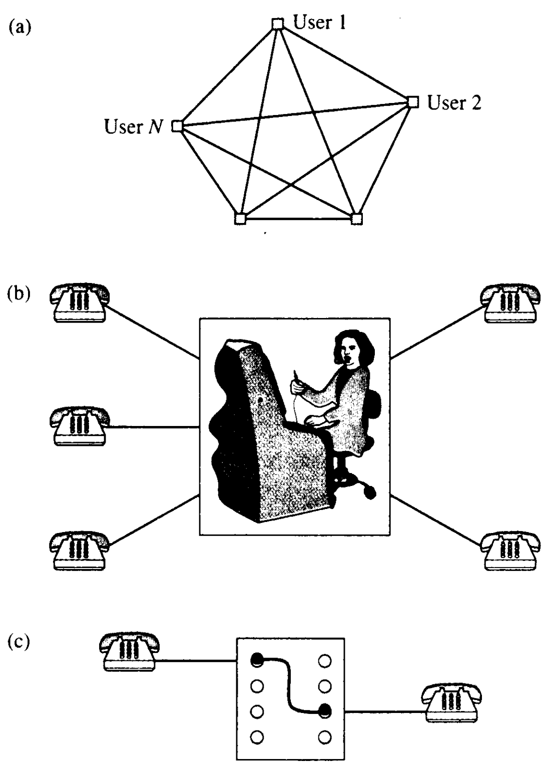


FIGURE 1.3 Telephone network: (a) dedicated resources require numerous lines; (b) a switch in the form of an operator with a patch cord panel; (c) cords interconnecting user sockets providing end-to-end connection.

In its simplest form, the telephone switch consists of a patch cord panel and a human operator as shown in Figure 1.3b. Traditionally a telephone call has three phases. In the first phase (the setup phase), the originating user picks up the telephone and in the process activates a signal in the circuit that connects it to the telephone office. The signal alerts the operator in a central office (CO) that a connection is requested. The operator speaks to the originating user and takes the requested destination station number and checks to see whether the desired user is available. If so, the operator establishes a connection by inserting the two ends of a cord into the sockets that terminate the lines of the two users as shown in Figure 1.3c. This connection allows electrical current, and the associated voice signal, to flow between the two users. If a telephone connection involves more than a single telephone office, then several operators collaborate to establish an end-to-end connection. This connection is maintained for the duration of the call. Once the connection is established the second phase (information transfer) begins. When the users are done with their conversation, they "hang up" their telephones, which generates a signal indicating that the call is complete. At this point, the third phase (connection release) is entered and the various telephone lines involved in the connection are then made available for new connections.

Starting in the 1890s with the invention of the Strowger switch, the patch panel switches and operators were replaced by automated electromechanical switches that could take a signal that contained the destination telephone number and automatically establish a circuit to the desired telephone. Telephone switching offices were

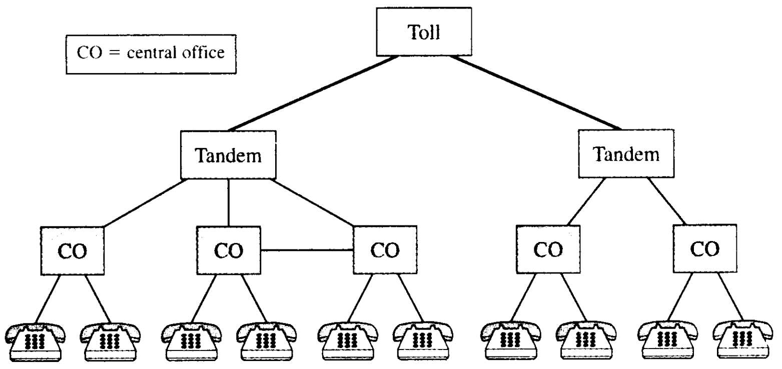


FIGURE 1.4 Hierarchical telephone network structure.

interconnected and organized in a hierarchical network, with customers attaching to end offices, which in turn are connected to tandem offices and so forth as shown in Figure 1.4. A hierarchical decimal **telephone numbering system** was developed for dialing connections in the telephone network. For example, in North America the area code specifies a subarea that has been assigned a three-digit number. The next three numbers are the exchange code, which identifies specific switching facilities in a central office within the subarea. The final four digits specify a specific line that connects the user to the central office. For example, the number 416-967-1111 will get you to a phone in Toronto (area code 416) where you can order pizza that will be delivered in 30 minutes or it's free.[3] Figure 1.5 shows how the telephone connection continues to consist of three phases: setup, transfer, and release.

We say that telephone networks are **connection-oriented** because they require the setting up of a connection before the actual transfer of information can take place. The transfer mode of a network that involves setting up a dedicated end-to-end connection is called **circuit switching.** Note that in circuit switching the **routing** decision is made when the path is set up in switching and transmission equipment across the network. After the call has been set up, information flows continuously across each switch and transmission line along the path. No additional address information is required after the call is set up.

The telephone network has undergone a gradual transition from analog technology to its present state, where it is almost completely based on digital transmission and computer technology. This transition began with the invention of the transistor in 1948 and accelerated with the invention of integrated circuits in the 1960s, leading to the development of digital transmission systems that could carry voice. In these systems an analog voice signal, such as the one shown in Figure 1.2, is converted into a binary stream of 64 kilobits/second (kbps) and the resultant digital signal is carried in a digital transmission system that interconnects two telephone offices. The T-1 digital transmission system was first deployed in 1962 to carry voice signals between telephone central offices. The T-1 system also provided for the **multiplexing** of 24 digitized voice signals

[3]Some restrictions apply!

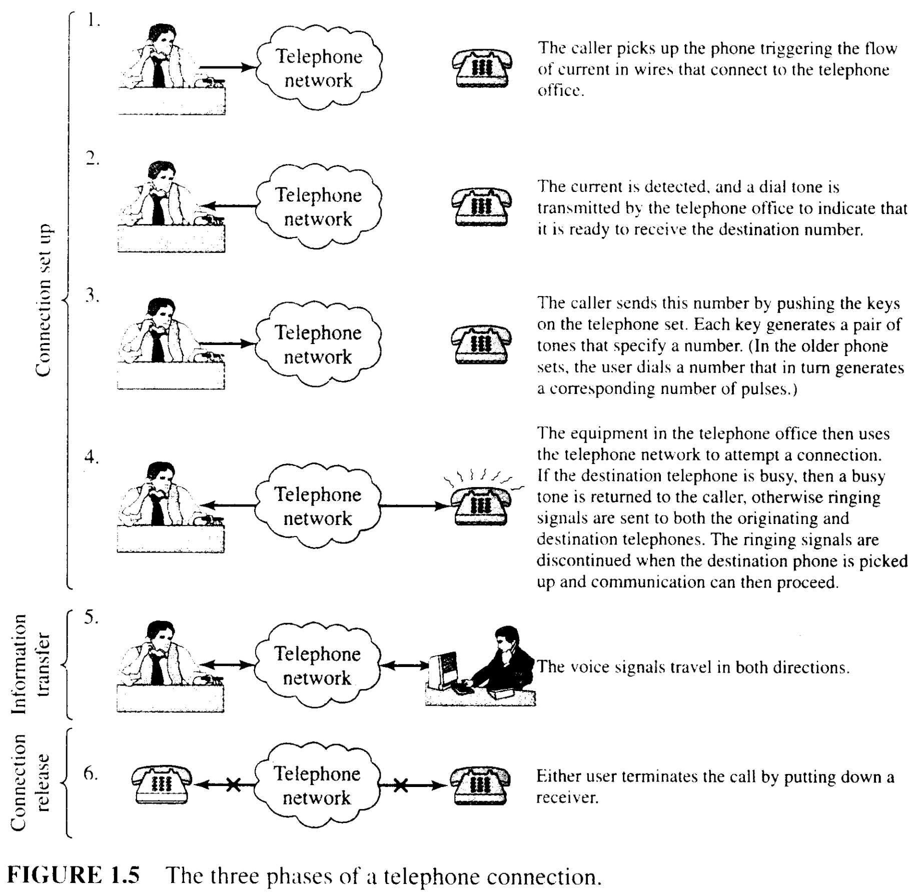

Connection set up
1.
Telephone network
The caller picks up the phone triggering the flow of current in wires that connect to the telephone office.
2.
Telephone network
The current is detected, and a dial tone is transmitted by the telephone office to indicate that it is ready to receive the destination number.
3.
Telephone network
The caller sends this number by pushing the keys on the telephone set. Each key generates a pair of tones that specify a number. (In the older phone sets, the user dials a number that in turn generates a corresponding number of pulses.)
4.
Telephone network
The equipment in the telephone office then uses the telephone network to attempt a connection. If the destination telephone is busy, then a busy tone is returned to the caller, otherwise ringing signals are sent to both the originating and destination telephones. The ringing signals are discontinued when the destination phone is picked up and communication can then proceed.
Information transfer
5.
Telephone network
The voice signals travel in both directions.
Connection release
6.
Telephone network
Either user terminates the call by putting down a receiver.

FIGURE 1.5 The three phases of a telephone connection.

into a single digital transmission line and operated at a transmission rate of 1.5 Mbps as shown in Figure 1.1.

Even as digital transmission systems were being deployed, the new digital transmission systems had to interface to existing analog switches. Upon arrival at an analog switch, the digital signal would be reconverted into analog voice signals for switching and then reconverted to digital form for transmission in the next hop. This inefficiency was eliminated by the deployment of *digital switches* in the early 1980s that can switch the voice signals in digital form. Thus a voice call would need to be digitized only once upon entering the network; then it would be transmitted and switched in digital form until it reached the other end of the network. The call would then be converted to analog form for transmission over the pair of wires that connects the user to the network.

In the 1960s and 1970s computer-based *connection control* was introduced for the setting up of connections in a switch. Computers would examine a request message for a call as it came in, check to see whether the destination was available, and if so, make the appropriate connection. The use of computers to control switches provided great flexibility in modifying the control of a connection and in introducing new features. It also led to the introduction of a separate **signaling network** to carry the messages between the computers that controlled these switches. In addition to setting up calls, the signaling network enabled the introduction of new services such as credit card calls, long-distance calls, 800 calls, call screening, voice mail, and many other services.

A major application of the signaling network is in cellular telephone service. This service requires that mobile users be provided with seamless radio connectivity even as they move from an area or cell covered by one antenna to an adjacent cell covered by a different antenna. The signaling network handles the messages that coordinate the handoff of users between cells. The usefulness of mobile communications led to explosive growth in the use of cellular phones. It also fostered new user expectations in terms of mobility and personal customization in their communications services not only for voice but also for data.

Let us now consider the interplay between telephone voice service and the architecture of the telephone network. Basic telephone voice service provides for the near-instantaneous ("real-time") transfer of voice signals between humans. The following elements and functions are essential in the architecture of the telephone network:

1. *Transmission systems* for the transfer of voice signals. These systems can be analog or digital depending on the format of the voice signal.
2. Telephone *switches* to transfer a signal from an incoming transmission line to an output transmission line. The switches can be analog or digital depending on the attached transmission system. The tandem connection of transmission lines and telephone switches form the end-to-end circuit that transfers the voice signal.
3. A telephone *numbering system* to identify telephone users or stations.
4. Methods for allowing users to indicate to the network that they require a connection, to specify the desired telephone number, and to indicate the termination of a call. We refer to this as a *user-to-network signaling system.*
5. A system inside the network, for performing a *routing* decision that identifies a path in the network, based on a request for a voice connection, and a *network signaling system* to distribute signaling messages to computers that control switches to establish the desired path between the two telephones.

As in telegraph networks, transmission systems form the foundation for the telephone network architecture. In addition, switches also play a role enabling the transfer of information from multiple users. In contrast to telegraph networks, telephone networks require that the routing elements identify and set up the entire path before any information flows into the network. Finally, it is noteworthy that the control part of the telephone network, that is, the signaling system, involves computer communications, that is, the exchange of messages between computers.

1.1.3 The Internet, Computer Networks, and Packet Switching

The Internet Protocol (IP) provides datagram service, namely, the transfer of "packets" of information across *multiple*, possibly dissimilar networks. Before discussing packet switching and the architecture of the Internet, we need to present some of the types of networks that the Internet operates on. We also retrace the development of computer network architectures.

The first computer network was the Semi-Automatic Ground Environment (SAGE) system developed between 1950 and 1956 for air defense systems [Green 1984]. The system consisted of 23 computer networks, each network connecting radar sites, ground-to-air data links, and other locations to a central computer. The SABRE airline reservations system, which was introduced in 1964, is cited as the first large successful commercial computer network and it incorporated many of the innovations of the SAGE system.

Early computers were extremely expensive, so techniques were developed to allow them to be shared by many users. In the 1960s tree-topology *terminal-oriented networks* were developed to allow user terminals to connect to a single central shared computer. As the cost of computers dropped and their use in organizations proliferated, it became necessary for users to connect to different computers for different applications. This situation required a different topology than that of terminal-oriented networks. In addition, as "dumb" terminals were replaced by "intelligent" terminals and later by personal computers, it became necessary to develop networks that were more flexible and could provide communications among many computers.

The ARPANET was the first major effort at developing a network to interconnect *computers* over a wide geographical area.[4] We emphasize the fact that the "users" of this network were full-fledged computers, not terminals. As such, the users of this network had processing and storage resources not available in previous terminal equipment. It therefore became possible to develop powerful networking protocols that made use of this processing capability at the edge of the network and to simplify the operation of the equipment inside the network. This approach is in marked contrast to the telephone network where the signaling and connection-control intelligence resides inside the network, not in the telephone set.

The Internet was later developed to enable communications between computers that were attached to different networks. IP made it possible to transfer information in the form of packets across many, dissimilar networks. In effect IP is used to create a single global internetwork out of many diverse networks. The TCP/IP protocols that emerged in the late 1970s form the basis for today's Internet.

WHAT IS A PROTOCOL?

In dealing with networks we run into a multiplicity of protocols, with acronyms such as HTTP, FTP, TCP, IP, DNS, and so on. What is a protocol, and why are there so many? A **protocol** is a set of rules that governs how two or more communicating parties are to interact.

As an example of a protocol, Figure 1.6 shows a typical exchange of messages that occurs when you call the telephone company to find a phone number. The interaction begins with the user dialing for directory assistance (411). The telephone network responds with an automated welcoming message followed by a request for the city (and state or province depending on the location of the city). Once the user

[4]The Advanced Research Projects Agency (ARPA) of the U.S. Department of Defense funded the development of the ARPANET.

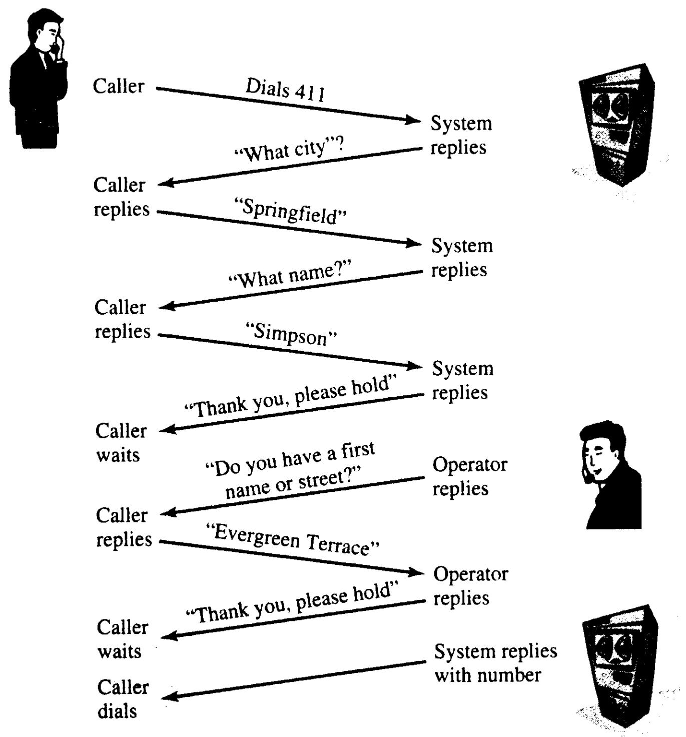


FIGURE 1.6 411 protocol: typical message exchange when requesting a telephone number.

responds, the network requests the name. After the user states the name, a human operator comes online and confirms the name and address. When the user provides the information, an automated message is played followed by a synthesized voice stating the phone number. The interaction or protocol between the caller and system can be defined quite precisely because the service provided is so specific.

In the case of computers, protocols must be precise and unambiguous as they involve communications between machines. For this reason, protocols involve defining the messages that are to be exchanged between machines as well as the actions that are to be taken when certain events take place and when certain messages are transmitted or received.

The purpose of a protocol is to provide some type of service. In web browsing the HyperText Transfer Protocol (HTTP) specifies how a web client and server are to interact. Other examples of protocols are: File Transfer Protocol (FTP) for the transfer of files. Simple Mail Transfer Protocol (SMTP) for e-mail, Internet Protocol (IP) for the transfer of packets, and Domain Name System (DNS) for IP address lookup. We discuss these protocols further in Chapter 2.

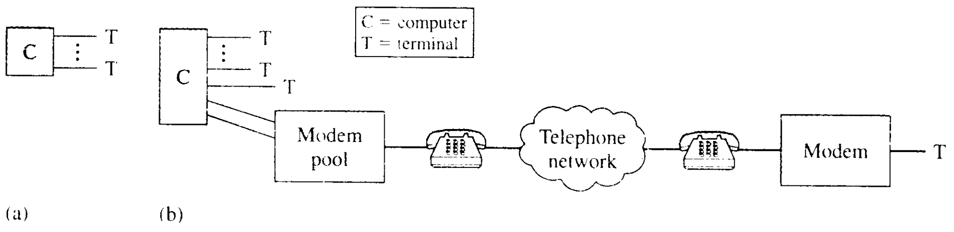


FIGURE 1.7 Terminal-oriented networks: (a) time-shared computers and cables for input devices; (b) dial-in.

TERMINAL-ORIENTED NETWORKS

Figure 1.7a shows an arrangement that allows a number of terminals to share a host computer. Each terminal, initially a teletype printer and later a video terminal, is connected by a set of wires to the computer. By exchanging messages with the host computer, the terminal could input instructions and obtain results from the computer. Initially the terminals were all located in a room adjacent to the host computer. Access from terminals located farther from the host computer became possible as communication lines with greater reach became available. *Modem* devices for transmitting digital information were introduced so that terminals could access the host computer via the telephone network, as shown in Figure 1.7b. Eventually terminal-oriented networks were developed to provide the *message transfer service* that enabled host computers to be shared.

Certain applications required a large number of geographically distributed terminals to be connected to a central computer. For example, a set of terminals at various travel agencies in a city might need to access the same computer. In most of these applications, the terminals would generate messages in a *bursty* manner; that is, the message transmissions would be separated by long idle times. The cost of providing individual lines to each terminal could be prohibitive, so various systems were developed to provide a message transfer service between the central computer and its associated terminals.

Medium access control methods were developed to allow a number of terminals at different locations to communicate with a central computer using a shared communication line. The access to the line needs to be controlled so that different terminals do not interfere with each other by transmitting at the same time. Figure 1.8 shows a "multidrop" line arrangement, allowing several terminals to share one line to and from the computer. This system uses a master/slave polling arrangement whereby the central computer sends a poll message to a specific terminal on the outgoing line. All terminals

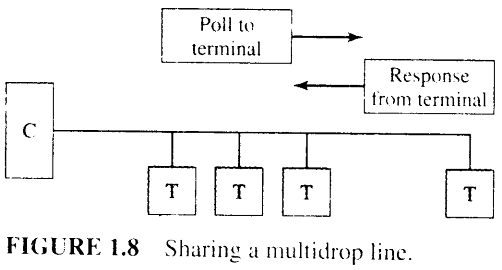


FIGURE 1.8 Sharing a multidrop line.

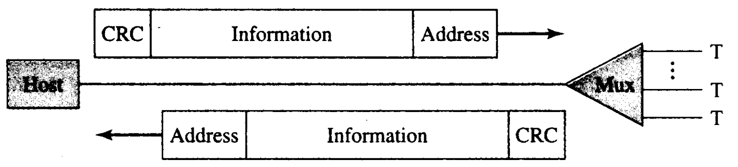


FIGURE 1.9 Multiplexer systems.

listen to the outgoing line, but only the terminal that is polled replies by sending any information that it has ready for transmission on the incoming line.

Statistical multiplexers/concentrators provided another means for sharing a communications line among terminals, as shown in Figure 1.9. Messages from a terminal are *encapsulated* inside a **frame** that consists of a **header** in addition to the user message. The header provides an **address** that identifies the terminal. The communication line transmits a sequence of binary digits, so a *framing* method is required to delineate the beginning and end of each frame. The messages from the various terminals are buffered by the multiplexer, ordered into a queue, and transmitted one at a time over the communication line to the central computer. The central computer sorts out the messages from each terminal, carries out the necessary processing, and returns the result inside a frame. The statistical multiplexer uses the address in the frame to determine the destination terminal.

Computer applications require precise instructions to operate correctly. Early data transmission systems that made use of telephone lines had to deal with errors in transmission arising from a variety of sources: interference from spurious external signals, impulses generated by analog switching equipment, and thermal noise inherent in electronic equipment. *Error-control* techniques were developed to ensure virtually error-free communication of data information. In addition to the header, each frame of information would have a number of redundant CRC or "check bits" appended to it prior to transmission as shown in Figure 1.9. These check bits were calculated from the information bits according to some algorithm and would enable the receiver to detect whether the received frames contained errors, and, if so, to request retransmission.

Figure 1.10 shows a typical terminal-oriented network circa 1970. Remote concentrators/statistical multiplexers at regional centers connect to a central computer, using high-speed digital transmission lines. Each remote concentrator gathers messages

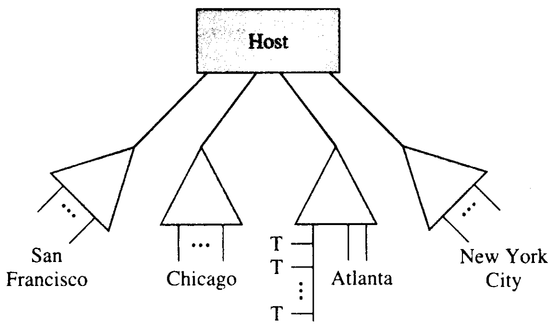


FIGURE 1.10 Typical terminal-oriented network circa 1970.

using lower-speed lines from various sites in its region. Multidrop lines such as those discussed above could be used in such lines, for example, Atlanta in the figure. Note that routing and forwarding are straightforward in these tree-topology networks as all information flows from terminals to the central host and back.

Let us now consider the architecture of terminal-oriented networks. The service provided by these networks involves the transfer of messages to and from a central computer to a set of terminals. The elements in the architecture of terminal-oriented networks is similar to that of telegraph networks:

1. A *digital transmission system* to transfer binary information. The transmission system can consist of a modem operating over telephone lines or of specialized data transmission equipment.
2. A method for the transmission of *frames* of information between terminals and central computers. The frames contain *address* information to identify the terminals, check bits to enable *error control,* and associated *framing* method to identify the boundaries of a frame.
3. In the case of systems that are connected to the same communication line, the systems also need a *medium access control* to coordinate the transmissions of information from terminals into the shared communication line.

COMPUTER-TO-COMPUTER NETWORKS

The basic service provided by computer networks is the transfer of messages from any computer connected to the network to any other computer connected to the network. This function is similar to the message switching service provided by telegraph systems. An additional requirement of a computer network is that transit times be short for messages from interactive applications. This requirement suggests that a limit be imposed on the size of messages that are allowed to enter the network, since long messages can impose long waiting times for interactive traffic that consists of short messages. **Packet-switching networks** address this problem. These networks are designed to provide *packet transfer service,* where a **packet** is a variable-length block of information up to some specified maximum size. User messages that do not fit inside a single packet are segmented and transmitted using multiple packets. As shown in Figure 1.11

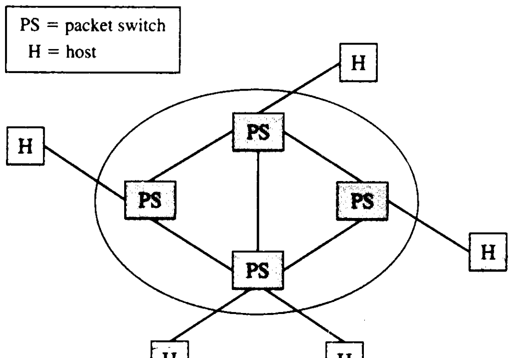


FIGURE 1.11 A packet-switching network.

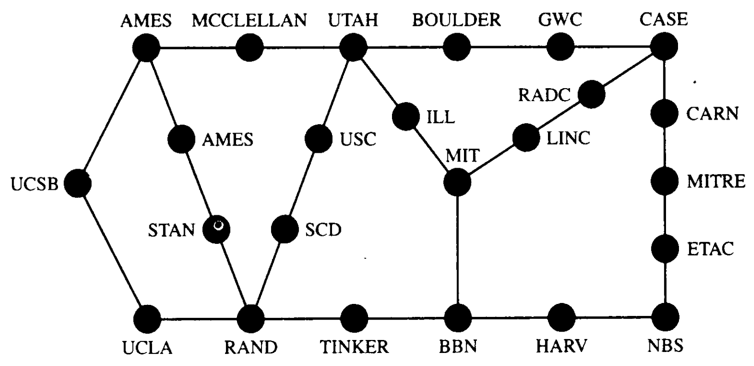


FIGURE 1.12 ARPANET circa 1972 [Kahn 1972].

packets are transferred from packet switch to packet switch until they are delivered at the destination. The messages are then recovered from individual packets or reassembled once the component packets have been received at the destination.

The ARPANET[5]

The **ARPANET** was developed in the late 1960s to provide a test bed for researching the interconnection of host computers using packet switching across a **wide area network (WAN),** a network that can span an area larger than a city and that can even be global in reach. From the attached host computer's point of view, the ARPANET offered a message transfer service. However, inside the network, the ARPANET operated on the basis of a packet transfer service. Each packet consists of a header with a destination address attached to user information and is transmitted as a single unit across a network, much as a telegram would be transmitted in a telegraph network. The ARPANET consisted of *packet switches* interconnected by communication lines that provided multiple paths for interconnecting host computers over wide geographical distances.[6] The packet switches were implemented by dedicated minicomputers, and each packet switch was connected to at least two other packet switches to provide alternative paths in case of failures. The communications lines were leased from public carriers and initially had a speed of 56 kbps. The resulting network had a topology, such as shown in Figure 1.12. In the ARPANET host computers exchanged messages of up 8000 bits in length, and the packet switching network was designed to transmit packets of information no longer than a given maximum length, about 1000 bits. Therefore, messages entering the network might require segmentation into one or more packets at the ingress packet switch and reassembly at the egress packet switch.

ARPANET packet transmission service was **connectionless** in the sense that no connection setup was required prior to the transmission of a packet. Thus packets could be transmitted immediately without incurring the delay associated with setting up a connection. Each packet contained *destination address information* that enabled the packet switches in the network to carry out the **routing** of the packet to the destination.

[5]See [Kahn 1972], also the website entitled "A Technical History of the ARPANET—A Technical Tour."
[6]In ARPANET a packet switch was called an IMP, for Interface Message Processor.

Each packet switch maintained a routing table that would specify the output line that was to be used for each given destination. Packets were then buffered to await transmission on the appropriate link. Packets from different users would thus be multiplexed into the links between packet switches.

Packets were encapsulated in *frames* that used special characters to delineate the frame, as well as check bits to enable *error control.* If errors were detected in an arriving frame, the receiving packet switch would discard the frame (and its packet). The sending packet switch would maintain a timer for each transmitted frame. A frame would be retransmitted if its timer expired before the frame was acknowledged.

Each packet switching node implemented a *distributed routing* algorithm to maintain its routing tables. The algorithm involved the exchange of information between neighboring nodes and the calculation of a consistent set of routing tables that together directed the flow of packets from source to destination. This arrangement enabled the routing tables to be updated in response to changes in traffic or topology and gave ARPANET the capability to adapt to faults in the network. The packets would simply be routed around the points of failure after the failures were detected and the routing tables updated. Because routes could change, it was also possible for packets to arrive out of order at the destination.

Each packet switch in ARPANET contained a limited amount of buffering for holding packets. To prevent packet switches from becoming congested, an end-to-end *congestion control* was developed. Operating between the source and destination packet switching nodes, the congestion control limited the number of packets that a host can have in transit.

In addition to defining the protocols for the operation of the packet network, the ARPANET also defined protocols for the interaction between host computers *across* the network. In particular, *flow control* methods were introduced to regulate the rate at which a source computer sent messages to a destination computer. This action prevented a sending computer from overflowing the buffers of a destination computer.

The motivation for developing computer networks was to support multiple user *applications.* The ARPANET developed several lasting applications that ran in its host computers. These included e-mail, remote login, and file transfer—applications that we take for granted today.

ARPANET was the first large-scale wide-area packet-switching network. ARPANET provided for the transfer of messages between host computers using a connectionless packet switching network. Its architecture incorporated most of the elements of modern packet switching networks:

1. High-speed (for the time) 56 kbps *digital transmission* lines provide for the transfer of streams of binary information.
2. Packets between adjacent packet switches are transferred inside of *frames* that contain delineation information as well as check bits for error control.
3. Destinations are identified by unique *addresses* that are used by routing tables to perform *routing* decisions to select the next hop of each packet as it traverses the network.
4. *Routing* tables at the packet switches are calculated dynamically in response to changes in network traffic and topology.

5. *Messages* are segmented into packets at the ingress to the network and reassembled at the egress of the network. End-to-end *congestion control* mechanisms are used to prevent congestion inside the network by limiting the number of packets a host can have in transit.
6. *Flow control* methods between host computers are introduced to prevent buffer overflow.
7. *Applications* that rely on the transfer of messages between computers are developed.

In the ARPANET we see most of the elements of computer network architecture.[7] Each element can be viewed as a *layer* in a vertical hierarchy or in an onionlike sphere. At the lowest layer we have digital transmission providing for the transfer of bits across a line. At each end of the line we have devices that exchange frames subject to rules that enable recovery from errors. Packet-switching nodes, interconnected by communications lines, use routing tables to direct packets from ingress nodes to egress nodes in the network. To determine the paths that should be built into the routing tables, all the packet-switching nodes run a distributed routing algorithm. Devices at the edge of the packet-switching network segment messages into packets and reassemble them back into messages. These devices may also regulate the rate at which packets are allowed into the network in order to prevent congestion inside the network. Host computers in turn may regulate the rate at which messages are sent to match the ability of the destination computer to process messages. The resulting computer network in turn enables the running of applications over geographically separate computers. In Chapter 2 we will examine the layered structure of network architectures in detail.

Local Area Networks

The emergence in the 1980s of workstations that provided computing at lower cost led to a proliferation of individual computers within a department or building. To minimize the overall system cost, it was desirable to share (then) expensive devices such as printers and disk drives. This practice gave rise to networks with limited distances (typically less than one kilometer) called **local area networks (LANs).** The requirements of a LAN are quite different from those of a wide area network. The small distances between computers in a LAN implied that low cost, very high speed, and relatively error-free communication was possible. Complex error-control procedures were largely unnecessary. In addition, in the local environment machines were constantly being moved between labs and offices, which created the administrative problem of keeping track of the location of a computer at any given time. This problem is easily overcome by giving the *network interface card* (*NIC*) for each machine a *globally unique address* and by **broadcasting** all messages to all machines in the LAN. A **medium access control** protocol becomes essential to coordinate access to the transmission medium in order to prevent collisions between frames. A variety of topologies can provide the broadcasting feature required by LANs, including ring, bus, and tree networks.

The most successful LAN, the **Ethernet** shown in Figure 1.13a, involved transmission over a bus topology coaxial cable. Stations with messages to transmit would first

[7]The missing element is that of an internetwork, or "internet."

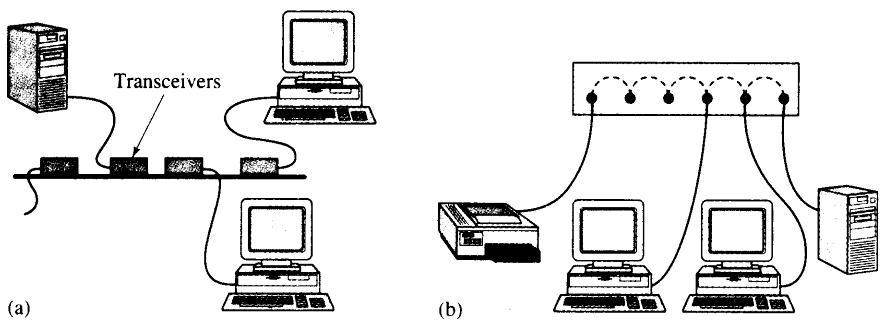


FIGURE 1.13 Ethernet local area network: (a) bus topology; (b) star topology.

sense the cable (carrier sensing) for the presence of ongoing transmissions. If no transmissions were found, the station would proceed to transmit its message encapsulated inside a **frame.** The station would continue to monitor the cable in an attempt to detect collisions. If collisions were detected, the station would abort its transmission. The introduction of sensing and collision detection significantly improved the efficiency of the transmission cable.

The bus topology of the original Ethernet has a disadvantage in terms of the cost of coaxial wiring relative to the cost of telephone cable wires as well as in terms of fault handling. Twisted-pair Ethernet was developed to provide lower cost through the use of conventional unshielded copper wires such as those used for telephones. As shown in Figure 1.13b, the computers are now connected by copper wires in a star topology to a hub that can be located in a wiring closet in the same way that telephones are connected in a building. The computers transmit packets using the same random access procedure as in the original Ethernet, except that collisions now occur at the hub, where the wires converge.

A LAN provides a message-transfer service between computers and other devices that are attached to the LAN. The elements of LAN architecture involve:

1. A high-speed (typically 10 megabits/second and above) *digital transmission* system that may support broadcast transmissions.
2. An *addressing* structure that provides each station with a unique address and supports broadcast transmissions.
3. A *frame* structure to delineate individual transmissions, and a *medium access control* procedure to coordinate transmissions into the shared broadcast medium.

The Internet

In the mid-1970s after the ARPANET packet-switching network had been established, ARPA began exploring data communications using satellite and mobile packet radio networks. The need to develop protocols to provide communications across multiple, possibly dissimilar, networks soon became apparent. An **internetwork** or **internet** involves the interconnection of multiple networks into a single large network, as shown in Figure 1.14. The component networks may differ in terms of their under-

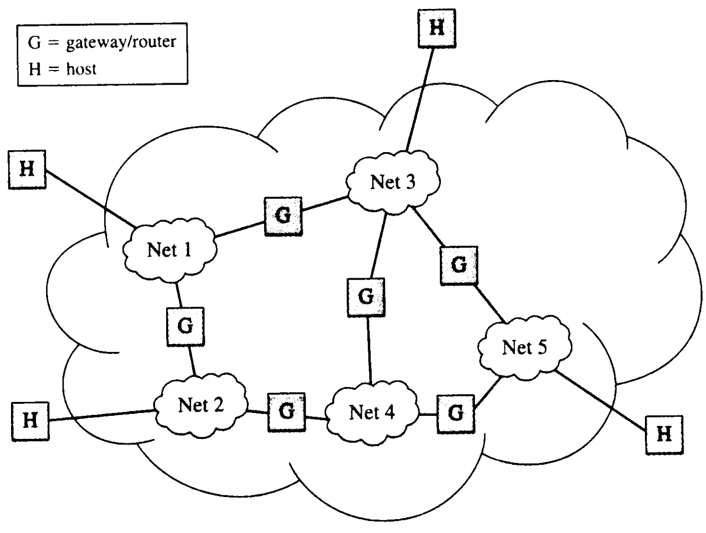


FIGURE 1.14 An internetwork.

lying technology and operation. For example, these networks could consist of various types of LANs, packet-switching networks, or even individual point-to-point links. The power in the internet concept is that it allows different networks to coexist and interwork effectively.

The **Internet Protocol (IP)** was developed to provide for the connectionless transfer packets called **datagrams** across an internetwork. In IP the component networks are interconnected by special packet switches called **gateways** or **routers.** Each router interface adapts to the particular attributes of the underlying network. IP routers direct the transfer of IP packets across an internet. After a routing decision is made, the packets are placed in a buffer to await transmission over the next network. In effect, packets from different users are statistically multiplexed in these buffers. The underlying networks are responsible for transferring the packets between routers.

IP traditionally provides **best-effort service.** That is, IP makes every effort to deliver the packets but takes no additional actions when packets are lost, corrupted, delivered out of order, or even misdelivered. In this sense the service provided by IP is unreliable. The student may wonder why one would want to build an internetwork to provide unreliable service. The reason is that providing reliability inside the internetwork introduces a great deal of complexity in the routers. The requirement that IP operate over any network places a premium on simplicity. For example, IP deals with congestion by dropping packets. This action triggers protocols in end systems at the edge of the network to adjust the rate at which they transmit.

IP uses a *hierarchical address space* that has "grouping" information embedded in the structure. IP addresses consist of 4 bytes usually expressed in dotted-decimal notation, for example, 128.100.11.1. IP addresses consist of two parts: a network ID and a host ID. Machines in the same group share common portions of the address, which allows routers to handle addresses with the same prefix in the same manner.

The Internet also provides a *name space* to refer to machines connected to the Internet, for example, tesla.comm.toronto.edu. The name space also has a hierarchical structure, but it is administrative and not used in the routing operation of the network.. Automatic translation of names to addresses is provided by the **Domain Name System (DNS).**

The **User Datagram Protocol (UDP)** allows applications to transfer individual blocks of user information using datagrams. UDP takes the user information, appends appropriate information that identifies the application in the destination host, and then uses IP to transfer the datagram across an internet. However, UDP is not appropriate for some applications. For example, many applications also require the reliable transfer of a stream of information in the correct sequence or order. The **Transmission Control Protocol (TCP)** was developed to provide reliable transfer of stream information over the connectionless IP. TCP operates in a pair of end hosts across an IP internet. TCP provides for error and flow control on an end-to-end basis that can deal with the problems that can arise due to lost, delayed, or misdelivered IP packets. TCP also includes a mechanism for reducing the rate at which information is transmitted into an internet when congestion is detected. TCP exemplifies the IP design principle, which is that complexity is relegated to the edge of the network, where it can be implemented in the host computers.

A rich set of applications has been developed to operate on top of the TCP/IP. These applications include SMTP for e-mail service, FTP for file transfer, HTTP for web service, and RTP for real-time transfer of information such as voice and video. By building on top of TCP/IP, applications are assured of two huge advantages: (1) the application will work over the entire Internet; (2) the application will continue working even as the underlying network technologies change and evolve because IP will continue to work on top of the new technologies.

In comparing the architecture of the Internet with that of ARPANET we see that IP introduces a layer between the computers that attach to it and individual packet networks. In effect, IP introduces a layer that creates a network of networks in which IP routers treat component networks as "links" that its packets must traverse. IP routers direct the flow of packets across networks. Devices attached to the Internet convert messages into packets and back. Indeed the key elements of the TCP/IP architecture are captured in the Federal Networking Council (www.itrd.gov/fnc/Internet_res.html) definition of the term **Internet** as a global information system that

a. *Is logically linked together by a global unique address space based on the Internet Protocol (IP) or its subsequent extensions/follow-ons.*
b. *Is able to support communications using the Transmission Control Protocol/Internet Protocol (TCP/IP) suite or its subsequent extenstions/follow-ons, and/or other IP-compatible protocols.*
c. *Provides, uses, or makes accessible, either publicly or privately, high-level services layered on the communications and related infrastructure described herein.*

Note that the above definition for Internet (with uppercase I) specifies *both* the use of TCP/IP protocols and the use of a globally unique address space. Thus a private internetwork that uses TCP/IP and a private address space is referred to as an internet (with lowercase i).

WHAT IS A DOMAIN NAME?

While each machine on a network has an IP address, this address usually provides little information on the use of the machine or who its owner is. To make the addresses more meaningful to people, host names were introduced. For example, the IP address 128.100.11.1 is not as informative as the corresponding domain name utoronto.ca, which of course identifies a host belonging to the University of Toronto.

Today host names are determined according to the *domain name system (DNS)*. The system uses a hierarchical tree topology to reflect different administrative levels. Below the root level are the familiar terms such as com, org, and edu as well as country identifiers, such as jp and ca, for Japan and Canada, respectively. When written linearly, the lowest level is the leftmost term. Thus, the domain name of the Network Architecture Lab at the University of Toronto is comm.utoronto.ca, where comm is a subdomain of utoronto.ca.

DNS is discussed further in Chapter 2.

ESSENTIAL ELEMENTS OF A NETWORK ARCHITECTURE

We have seen in this section that certain elements or functions are essential to achieve communications, hence their presence in most modern network architectures. These elements include the following:

1. *Digital transmission* lines for the transfer of streams of binary information between equipment.
2. Exchange of *frames* of information between adjacent equipment; these frames contain delineation information as well as check bits for error control.
3. *Medium access control* procedures to regulate the transmission of frames from multiple users to a shared broadcast medium.
4. *Addresses* to identify points of attachment to a network or internetwork.
5. Exchange of *packets* of information between packet switches in a network. *Routing* tables in packet switches are used to select the path of each packet as it traverses the network.
6. *Dynamic calculation* of routing tables at the packet switches in response to changes in network traffic and topology.
7. *Congestion control* mechanisms may be used to prevent congestion inside the network.
8. *Internetworking* provides connectivity across multiple, possibly dissimilar, networks by using gateways or routers.
9. *Segmentation and reassembly* of messages into packets at the ingress to and egress from the network. *End-to-end error recovery* mechanisms to ensure reliable transfer across a network or internetwork.
10. A *multiplicity of applications* that build on the transfer of messages between computers.

The above network elements can be organized in a layered fashion so that one builds on top of another. For example, the framing element builds on top of a digital transmission to enable the transfer of blocks of information across a transmission line. Another example involves an application building on top of a network information service. For example, the HTTP application builds on top of TCP/IP to deliver the web-browsing service. In Chapter 2 we develop the notion of layered network architectures in detail.

We also saw that there are two fundamentally different approaches to communications. In the *connectionless* approach, the message is sent directly into the network, and no connection needs to be set up ahead of time. Telegrams worked in this fashion, and datagrams in computer networks work this way. The *connection-oriented* approach requires that a connection be set up across equipment in the network before information can be transferred. The telephone network provides an important example of this approach. Interestingly the setting up of a connection involves the exchange of messages between controllers and switches and hence gives rise to the need for a computer network to carry the signaling messages. The contrast between connectionless and connection-oriented networks will be a recurring theme in the development of network architectures and we will discuss the issue throughout this book.

1.2 FUTURE NETWORK ARCHITECTURES AND THEIR SERVICES

We have summarized two centuries in the evolution of communication networks and services and in doing so have seen that new network architectures can emerge and obliterate old architectures. For example, the introduction of the electrical telegraph led to the disappearance of the visual telegraph networks within a few years. We have also seen that a new architecture can grow steadily over many decades eventually displacing an older architecture. For example, the gradual global deployment of the telephone network eventually led to the disappearance of telegraph networks. We are currently experiencing a transition from a telephone-centric network architecture to a computer-centric architecture. In this section we examine some of the factors driving this transition.

The introduction of new applications and services is an important factor in network evolution. Typically, many new applications and services over time build on top of the basic network service. Facsimile transmissions of documents and data communications using modems are two examples of services that build on top of the basic voice transmission service. Other new services have resulted from leveraging the capabilities of the telephone-signaling network. Examples include 1-800 calls, caller-id, call forwarding, and voice mail. Another very important example is provided by cellular telephone services, which use the telephone-signaling infrastructure to control the connections between users as they move within a geographical area.

The Internet has a network architecture that supports a relatively simple, but very powerful service, the transfer of packets across many networks. This basic service allows computers to exchange messages, and in so doing, enables *any* application that can be implemented through the exchange of messages between computers. Applications that are taken for granted today, e-mail for example, came into wide use only in the recent past because of the Internet. Other applications, such as the World Wide Web and Napster, underwent explosive growth and had dramatic impact on business, commerce, education, and entertainment. We can expect that new generations of applications will continue to emerge, and if found valuable, deploy rapidly in the Internet.

Applications and services that incorporate multimedia information, that is, audio, images, and video are an important factor affecting network evolution today. The transition from analog to digital formats for multimedia information is nearly complete: phonograph records and audiocassettes have been replaced by audio compact disks (CDs), film cameras are being replaced by digital cameras, and analog videocassette recorders and televisions are being replaced by digital versatile disk (DVD) recordings and MPEG digital television. The transition from networks that transfer multimedia information in analog form to networks that use digital means is also underway. Digital television is now distributed over cable and satellite television broadcast networks. The exchange of JPEG digital images and MP3 digital audio over the Internet is now commonplace. However, *neither* the telephone nor the Internet is capable of supporting the *real-time* transfer of multimedia information with high quality. This fact points to limitations in both architectures.

The circuit-switched telephone network was designed to provide real-time voice communications using pre-established connections. A *physical* circuit that bears the voice signal in electrical form provides as instantaneous transfer as is possible, namely, at the speed of light. The telephone network can deliver voice signals of excellent quality across the globe. Unfortunately the telephone network is not very flexible when it comes to handling other types of multimedia information. Specialized terminal equipment is needed to digitize audio-visual signals and to fit them into formats that are multiples of the 64 kbps rate of a single digital voice signal. In addition, special handling, and hence higher cost, is involved in setting up connections for such signals across the telephone network. For these reasons video-telephone service has never taken off.

Packet-switched networks have advantages as well as serious shortcomings in the handling of real-time multimedia information. Packet networks are not constrained to fixed bit-rate signals; any signal, whether audio, video, or something else, can be digitized and packed into packets for transfer across the network. However, packet networks can involve multiple, and usually variable, packet delays. Before a signal is delivered delays are incurred in digitization, in filling a packet with the digitized signal, and waiting for transmission in buffers at the packet switches in the path along the network. Best-effort packet transfer does not distinguish between packets from a real-time voice signal and packets from a file transfer, and so, voice packets can encounter very large delays making real-time packet communications of voice and other multimedia information impossible. We will see later in the book that current changes are being introduced in the Internet architecture so that packets can be handled with different levels of urgency and care. The support of *real-time* communications over the Internet will lead not only to voice-over-Internet service, but will spur many interesting new applications, from network games to real-time remote control of many types.

The confluence of telephone networks and the Internet is leading to new network architectures that combine elements from packet and circuit switching. The Asynchronous Transfer Mode (ATM) network architecture is a major example of the combination of packet and circuit-switching concepts. ATM networks enable the network operator to set up pipes that can carry either fixed or variable bandwidth traffic and to provide delay guarantees. New protocols in Internet provide means for establishing paths for IP *packet* traffic to provide delay guarantees as well as to manage traffic flows in packet networks. These new protocols are also influencing circuit-switching networks. Optical networks that provide very high bandwidth connections between machines such as routers operate in circuit-switched mode. In optical networks the distributed routing methods of IP networks are being adapted to provide a means for selecting optical paths in circuit-switched optical networks. Another major area of activity involves the development of signaling methods for use in the Internet to provide the call services available in the telephone network over the Internet, that is, voice telephony with call forwarding, etc., as well as the design of gateways to enable the interworking of telephone services across both networks. An especially significant application of these new capabilities is to provide mobility and personal customization in communication services. We examine these emerging protocols in the later part of this book.

Clearly network architectures are in a state of flux, and the reader can expect that the next few years will be very interesting.

1.3 KEY FACTORS IN COMMUNICATION NETWORK EVOLUTION

We have traced the evolution of communication networks from telegraphy to the emerging integrated services networks. Before proceeding with the technical details of networking in the remainder of the book, however, we pause to discuss factors that influence the evolution of communication networks. Figure 1.15 shows the three traditional factors: technology, regulation, and market. To these we add standards, a set of technical specifications followed by manufacturers or service providers, as a fourth factor.

A traditional axiom of telecommunications was that a new telecommunications service could succeed only if three conditions were satisfied. First of all the technology must be available to implement the service in a cost-effective manner. Second, government regulations must permit such a service to be offered. Third, the market

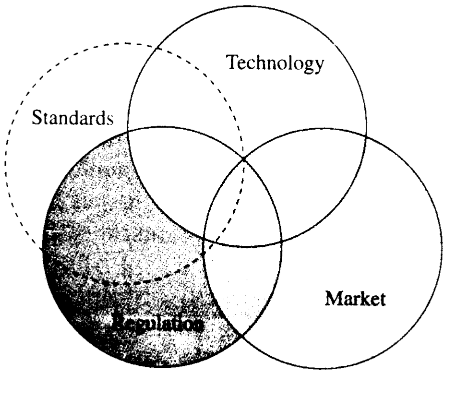


FIGURE 1.15 Factors determining success of a new service.

for the service must exist. These three conditions were applicable in the monopoly environment where a single provider made all the decisions regarding design and implementation of the network. The move away from single providers of network services and manufacturers of equipment made compliance with recognized standards essential.

The availability of the technology to implement a service in and of itself does not guarantee its success. Numerous failures in new service offerings can be traced back to the nontechnology factors. Frequently new services fall in gray areas where the regulatory constraints are not clear. For example, most regulatory policies regarding television broadcasting are intended for radio broadcast and cable systems; however, it is not clear that these regulations apply to television over the Internet. Also, it is seldom clear ahead of time that a market exists for a given new service. For example, the deployment of videotelephony has met with failure several times in the past few decades due to lack of market.

1.3.1 Role of Technology

Technology always plays a role in determining what can be built. The capabilities of various technologies have improved dramatically over the past two centuries. These improvements in capabilities have been accompanied by reductions in cost. As a result, many systems that were simply impossible two decades ago have become not only feasible but also cost-effective.

Of course, fundamental physical considerations place limits on what technology can ultimately achieve. For example, no signal can propagate faster than the speed of light, and hence there is a minimum delay or latency in the transfer of a message between two points a certain distance apart. However, while bounded by physical laws, substantial opportunities for further improvement in enabling technologies remain.

The capabilities of a given technology can be traced over a period of time and found to form an S-shaped curve, as shown in Figure 1.16a. During the initial phase the capabilities of the technology improve dramatically, but eventually the capabilities saturate as they approach fundamental limitations. An example of this situation is the capability

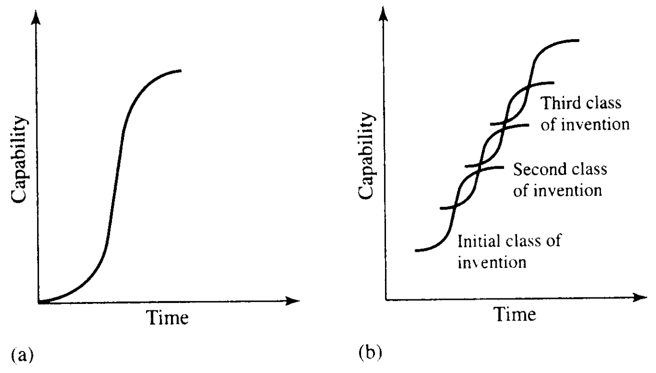


FIGURE 1.16 Capability of a technology as an S curve (based on Martin 1977).

of copper wires to carry information measured in bits per second. As the capabilities of a given technology approach saturation, innovations that provide the same capabilities but within a new technology class arise. For example, as copper wire transmission approached its fundamental limitation, the class of coaxial cable transmission emerged, which in turn was replaced by the class of optical fiber transmission. The optical fiber class has much higher fundamental limits in terms of achievable transmission rates and its S curve is now only in the early phase. When the S curves for different classes of technologies are superimposed, they form a smooth S curve themselves, as shown in Figure 1.16b.

In discussing the evolution of network architecture in Section 1.1, we referred to the rate curve for information transmission shown in Figure 1.1. The figure traces the evolution from telegraphy to analog telephony, computer networks, digital telephony, and the currently emerging integrated services networks. In the figure we also note various milestones in the evolution of networking concepts. Early telegraphy systems operated at a speed equivalent to tens of bits per second. Early digital telephone systems handled 24 voice channels per wire, equivalent to about 1,500,000 bits per second. In 1997 optical transmissions systems could handle about 500,000 simultaneous voice channels, equivalent to about 10^{10} bits per second (10 gigabits per second)! In the year 2000 systems operated at rates of 10^{12} bits per second (1 terabit per second) and higher! These dramatic improvements in transmission capability have driven the evolution of networks from telegraphy messaging to voice telephony and currently to image and video communications.

In addition to information transmission capacity, a number of other key technologies have participated in the development of communication networks. These include signal processing technology and digital computer technology. In particular, computer memory capacity and computer processing capacity play a key role in the operation of network switches and the implementation of network protocols. These two technologies have thus greatly influenced the development of networks. For more than three decades now, computer technology has improved at a rate that every 18 to 24 months the same dollar buys twice the performance in computer processing, computer storage, and transmission capacity. For example, *Moore's Law* states that the number of transistors per chip doubles every 1.5 years. Figure 1.17 shows the improvement in Intel microprocessors over the past three decades. These improvements have resulted in networks

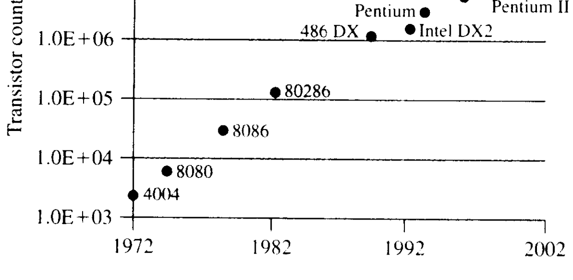


FIGURE 1.17 Moore's Law: The number of transistors per chip doubles every 1.5 years.

that not only handle greater volumes of information and greater data rates but can also carry out more sophisticated processing and hence support a wider range of services.

The advances in core technologies in the form of higher transmission, storage, and processing capacities are enablers to *bigger and more complex systems*. These advances enable the implementation and deployment of more intelligent, software-based algorithms to control and manage networks of increasingly larger scale. Major advances in software technologies have been needed to carry out the design, development, and testing, as well as to ensure the reliable and continued operation of these extremely complex systems.

1.3.2 Role of Regulation

Traditional communication services in the form of telephony and telegraphy have been government regulated. Because of the high cost in deploying the requisite infrastructure and the importance of controlling communications, governments often chose to operate communication networks as monopolies. The planning of communication networks was done over time horizons spanning several decades. This planning accounted for providing a very small set of well-defined communication services, for example, telegraph and "plain-old telephone service" (POTS). These traditional telegraph and telephone organizations were consequently not very well prepared to introduce new services at a fast rate.

The last three decades have seen a marked move away from monopoly environment for communications in North America, and to a certain extent in other parts of the world. The Carterfone decision by the U.S. Federal Communications Commission (FCC) in 1968 opened the door for the connection of non-telephone-company telephone equipment to the telephone network. The breakup in 1984 of the AT&T system into an independent long-distance carrier and a number of independent regional telephone operating companies opened the way for competition. In the United States the Telecommunications Act of 1996 attempted to open the way to further competition in the access portion of the telephone network. A surge in new entrants to local telephone access led to a boom in the growth of the telecom service and equipment industry. This "telecom bubble" burst in 2001 and led to the catastrophic demise of most of the new service providers. It is likely that some form of re-regulation will be reintroduced in the United States in the coming years.

In spite of the trend towards deregulation, telecommunications will probably never be entirely free of government regulation. For example, telephone service is now considered an essential "lifeline" service in many countries, and regulation plays a role in ensuring that access to a minimal level of service is available to everybody. Regulation can also play a role in addressing the issue of which information should be available to people over a communications network. For example, many people agree that some measures should be available to prevent children from accessing pornography over the Internet. However, there is less agreement on the degree to which information should be kept private when transmitted over a network. Should encryption be so secure that no one, not even the government in matters of national security, can decipher transmitted information? These questions are not easily answered. The point here is that regulation on these matters will provide a framework that determines what types of services and networks can be implemented.

1.3.3 Role of the Market

The existence of a market for a new service is the third factor involved in determining the success of a new service. This success is ultimately determined by a customer's willingness to pay, which, of course, depends on the cost, usefulness, and appeal of the service. *For a network-based service, the usefulness of the service frequently depends on there being a critical mass of subscribers.* For example, telephone or e-mail service is of limited use if the number of reachable destinations is small. In addition, the cost of a service generally decreases with the size of the subscriber base due *to economies of scale*, for example, the cost of terminal devices and their components. The challenge then is how to manage the deployment of a service to first address a critical mass and then to grow to large scale.

As examples, we will cite one instance where the deployment to large scale failed and another where it succeeded. In the early 1970s a great amount of investment was made in the United States in developing the Picturephone service, which would provide audiovisual communications. The market for such a service did not materialize. Subsequent attempts have also failed, and only recently are we starting to see the availability of such a service piggybacking on the wide availability of personal computers.

As a second example, we consider the deployment of cellular radio telephony. The service, first introduced in the late 1970s, was initially deployed as a high-end service that would appeal to a relatively narrow segment of people who had to communicate while on the move. This deployment successfully established the initial market. The utility of being able to communicate while on the move had such broad appeal that the service mushroomed over a very short period of time. The explosive growth in the number of cellular telephone subscribers prompted the deployment of new wireless technologies.

1.3.4 Role of Standards

Standards are basically agreements, with industrywide, national, and possibly international scope, that allow equipment manufactured by different vendors to be interoperable. Standards focus on interfaces that specify how equipment is physically interconnected and what procedures are used to operate across different equipment. Standards applying to data communications between computers specify the hardware and software procedures through which computers can correctly and reliably "talk to one another." Standards are extremely important in communications where the value of a network is to a large extent determined by the size of the community that can be reached. In addition, the investment required in telecommunications networks is very high, and so network operators are particularly interested in having the choice of buying equipment from multiple, competing suppliers, rather than being committed to buying equipment from a single supplier.

Standards can arise in a number of ways. In the strict sense, *de jure* standards result from a consultative process that occurs on a national and possibly international basis. For example, many communication standards, especially for telephony, are developed by the **International Telecommunications Union (ITU),** which is an organization that operates under the auspices of the United Nations.[8] Almost every country has its own corresponding organization that is charged with the task of setting national communication standards. In addition, some standards are set by nongovernmental organizations. Prominent examples include the **Internet Engineering Task Force (IETF),** which is responsible for the development of standards related to the Internet and the **Institute of Electrical and Electronic Engineers (IEEE) 802** committee, which specializes in LAN/MAN[9] standards. While the ITU follows a more formal standardization process, the IETF adopts a more pragmatic approach that relies on "rough consensus and running code." This key difference historically has led vendors to produce interoperable implementations based on IETF specifications faster. Standards may arise when a certain product, or class of products, becomes dominant in a market. An example of these *de facto* standards are personal computers based on Intel microprocessors and the Microsoft™ Windows operating system.

The existence of standards enables smaller companies to enter large markets such as communication networks. These companies can focus on the development of limited but key products that are guaranteed to operate within the overall network, for example, chips that implement certain protocols. This environment results in an increased rate of innovation and evolution of both the technology and the standards.

On a more fundamental level, standards provide a framework that can guide the decentralized activities of the various commercial, industrial, and governmental organizations involved in the development and evolution of networks.

CHECKLIST OF IMPORTANT TERMS

address
analog transmission system
ARPANET
best-effort service
broadcasting
circuit switching
communication network
connection-oriented
connectionless packet transfer
datagram
digital transmission system
Domain Name System (DNS)
Ethernet
frame
gateway/router
header
Institute of Electrical and Electronic Engineers (IEEE)
International Telecommunications Union (ITU)
Internet Engineering Task Force (IETF)
Internet
internetwork or internet
Internet Protocol (IP)

[8]The ITU was formerly known as the Consultative Committee for International Telegraphy and Telephony (CCITT).

[9]A metropolitan area network (MAN) typically covers an area of a city.

local area network (LAN)
medium access control
message switching
multiplexing
packet
packet switching
routing
services
signaling network
statistical multiplexer/concentrator
store-and-forward
telephone numbering system
telephone service
telephone switch
Transmission Control Protocol (TCP)
transmission rate
User Datagram Protocol (UDP)
wide area network (WAN)

FURTHER READING

"100 Years of Communications Progress," *IEEE Communications Magazine,* Vol. 22, No. 5, May 1984. Contains many excellent articles on the history of communications and predictions on the future of networks.

Bylanski, P. and D. G. W. Ingram, *Digital Transmission Systems,* Peter Peregrinus Ltd., England, 1980. Interesting discussion on Baudot telegraph system.

Carne, E. B., *Telecommunications Primer: Signals, Building Blocks, and Networks,* Prentice-Hall, Englewood Cliffs, New Jersey, 1995. Excellent introduction to telephone networks.

Davies, D. W., D. L. A. Barber, W. L. Price, and C. M. Solomonides, *Computer Networks and Their Protocols,* John Wiley & Sons, New York, 1979. Presents the state of the art in packet switching in the mid-1970s.

Goralski, W. J., *Introduction to ATM Networking,* McGraw-Hill, New York, 1996. Presents an interesting discussion on the history of telegraphy and telephony.

Green, P. E., "Computer Communications: Milestones and Prophecies," *IEEE Communications Magazine,* May 1984, pp. 49–63.

Kahn, R. E., "Resource-Sharing Computer Communication Networks," *Proceedings of the IEEE,* November 1972, pp. 1397–1407.

Leiner, B. M., V. G. Cerf, D. D. Clark, R. E. Kahn, L. Kleinrock, D. C. Lynch, J. Postel, L. G. Roberts, and S. Wolff, "A Brief History of the Internet," http://www.isoc.org/internet-history/brief.html, May 1997.

Martin, J., *Future Developments in Telecommunications,* Prentice-Hall, Englewood Cliffs, New Jersey, 1977. Details Martin's vision of the future of networking; interesting to look back and see how often his predictions were on target.

Perlman, R., *Interconnections: Bridges, Routers, Switches, and Internet Protocols,* Addison-Wesley, Reading, Massachusetts, 2000. Excellent discussion of fundamental networking principles.

Schwartz, M., R. R. Boorstyn, and R. L. Pickholtz, "Terminal-Oriented Computer-Communication Networks," *Computer Networks: A Tutorial,* M. Abrams, R. P. Blanc, and I. W. Cotton, eds., IEEE Press, 1978, pp. 2-18–2-33.

Stumpers, F. L. H. M., "The History, Development, and Future of Telecommunications in Europe," *IEEE Communications Magazine,* May 1984, pp. 84–95.

RFC1160, V. Cerf, "The Internet Activities Board," May 1990.

See our website for additional references available through the Internet.

PROBLEMS

1.1. (a) Describe the step-by-step procedure that is involved from the time you deposit a letter in a mailbox to the time the letter is delivered to its destination. What role do names, addresses, and mail codes (such as ZIP codes or postal codes) play? How might the letter be routed to its destination? To what extent can the process be automated?
(b) Repeat part (a) for an e-mail message. At this point you may have to conjecture different approaches about what goes on inside the computer network.
(c) Are the procedures in parts (a) and (b) connection-oriented or connectionless?

1.2. (a) Describe what step-by-step procedure might be involved inside the network in making a telephone connection.
(b) Now consider a personal communication service that provides a user with a personal telephone number. When the number is dialed, the network establishes a connection to wherever the user is located at the given time. What functions must the network now perform to implement this service?

1.3. Explain how the telephone network might modify the way calls are handled to provide the following services:
(a) Call display: the number and/or name of the calling party is listed on a screen before the call is answered.
(b) Call waiting: a special sound is heard when the called party is on the line and another user is trying to reach the called party.
(c) Call answer: if the called party is busy or after the phone rings a prescribed number of times, the network gives the caller the option of leaving a voice message.
(d) Three-way calling: allows a user to talk with two other people at the same time.

1.4. (a) Suppose that the letter in problem 1.1 is sent by fax. Is this mode of communications connectionless or connection-oriented? real-time or non real-time?
(b) Repeat part (a) for a voice-mail message left at a given telephone.

1.5. Suppose that network addresses are scarce and are assigned so that they are not globally unique; in particular, suppose that the same block of addresses may be assigned to multiple organizations. How can the organizations use these addresses? Can users from two such organizations communicate with each other?

1.6. (a) Describe the similarities and differences in the services provided by (1) a music program delivered over broadcast radio and (2) music delivered by a dedicated CD player.
(b) Describe how these services might be provided and enhanced by providing them through a communications network.

1.7. (a) Use the World Wide Web to visit the sites of several major newspapers. How are these newspapers changing the manner in which they deliver news services?
(b) Now visit the websites of several major television networks. How are they changing the manner in which they deliver news over the Internet? What differences, if any, exist between the approaches taken by television networks and newspapers?

1.8. Discuss the advantages and disadvantages of transmitting fax messages over the Internet instead of the telephone network.

1.9. (a) Suppose that an interactive video game is accessed over a communication network. What requirements are imposed on the network if the network is connection-oriented? connectionless?
(b) Repeat part (a) if the game involves several players located at different sites.
(c) Repeat part (b) if one or more of the players is in motion, for example, kids in the back of the van during a summer trip.

1.10. Discuss the similarities between the following national transportation networks and a communications network. Is the transportation system more similar to a telephone network or to a packet network?
(a) Railroad network.
(b) Airline network.
(c) Highway system.
(d) Combination of (a), (b), and (c).

1.11. In the 1950s standard containers were developed for the transportation of goods. These standard containers could fit on a train car, on a truck, or in specially designed container ships. The standard size of the containers makes it possible to load and unload them much more quickly than non-standard containers of different sizes. Draw an analogy to packet-switching communications networks. In your answer identify what might constitute a container and speculate on the advantages that may come from standard-size information containers.

1.12. The requirements of world commerce led to the building of the Suez and Panama Canals. What analogous situations might arise in communication networks?

1.13. Two musicians located in different cities want to have a jam session over a communication network. Find the maximum possible distance between the musicians if they are to interact in real-time, in the sense of experiencing the same delay in hearing each other as if they were 10 meters apart. The speed of sound is approximately 330 meters/second, and assume that the network transmits the sound at the speed of light in cable, 2.3×10^8 meters/second.

1.14. The propagation delay is the time required for the energy of a signal to propagate from one point to another.
(a) Find the propagation delay for a signal traversing the following networks at the speed of light in cable (2.3×10^8 meters/second):

a circuit board	10 cm
a room	10 m
a building	100 m
a metropolitan area	100 km
a continent	5000 km
up and down to a geostationary satellite	$2 \times 36{,}000$ km

(b) How many bits are in transit during the propagation delay in the above cases if bits are entering the above networks at the following transmission speeds: 10,000 bits/second; 1 megabit/second; 100 megabits/second; 10 gigabits/second.

1.15. In problem 1.14, how long does it take to send an L-byte file and to receive a 1-byte acknowledgment back? Let $L = 1, 10^3, 10^6$, and 10^9 bytes.

1.16. Use your web browser to access a search engine and retrieve the article "A Brief History of the Internet" by Leiner, Cerf, Clark, Kahn, Kleinrock, Lynch, Postel, Roberts, and Wolff. Answer the following questions:

(a) Who was J. Licklider, and what was his "galactic network" concept?
(b) Who coined the term *packet?*
(c) What (who?) is an IMP?
(d) Did the ARPANET use NCP or TCP/IP?
(e) Was packet voice proposed as an early application for Internet?
(f) How many networks did the initial IP address provide for?

1.17. Use your web browser to access a search engine and retrieve the following presentation from the ACM 97 conference: "The Folly Laws of Predictions 1.0" by Gordon Bell. Answer the following questions:

(a) At what rate have processing, storage, and backbone technologies improved from 1950 to 2000? How does this rate compare to advances in telephony?
(b) What is Moore's Law?
(c) What's the point of making predictions?
(d) What is the difficulty in anticipating trends that have exponential growth?
(e) Who was Vannevar Bush, and why is he famous?
(f) What is the size in bytes of each frame in this presentation? What is the size in bytes of the audio clip for a typical frame? What is the size of the video clip for a typical scene?

1.18. The introduction of new communications services usually impacts other services through substitution. Describe how substitution takes place in the following cases.

(a) E-mail, facsimile, and postal mail.
(b) E-mail, local, and long-distance phone service.
(c) Cell phone, local, and long-distance phone service.
(d) Peer-to-peer file exchange and commercial CD recording.

1.19. Use your web browser to access a news website and play a news video clip. Speculate about how the information is being transported over the Internet. How does the quality of the audio and video compare to that of broadcast or cable television?

1.20. Use your web browser to access the IETF web page at (currently at www.ietf.org) and learn the Internet Standards Process documented in RFC 2026.

(a) What are the different types of Requests for Comments (RFCs)?
(b) What is an Internet Draft?
(c) What are the differences among Proposed Standard, Draft Standard, and Standard?
(d) Which group in the IETF approves a certain specification for standard-track?
(e) How are disputes on working group documents resolved?

CHAPTER 2

Applications and Layered Architectures

architecture, *n. Any design or orderly arrangement perceived by man.*
design, *n. The invention and disposition of the forms, parts, or details of something according to a plan.*[1]

Communication networks can be called upon to support an extremely wide range of services. We routinely use networks to talk to people, to send e-mail, to transfer files, and to retrieve information. Business and industry use networks to carry out critical functions, such as the transfer of funds and the automated processing of transactions, and to query or update database information. Increasingly, the Internet is also being used to provide "broadcast" services along the lines of traditional radio and television. It is clear then that the network must be designed so that it has the flexibility to provide support for current services and to accommodate future services. To achieve this flexibility, *an overall network architecture or plan is necessary.*

The overall process of enabling two or more devices to communicate effectively across a network is extremely complex. In Chapter 1 we identified the many elements of a network that are required to enable effective communication. Early network designers recognized the need to develop architectures that would provide a structure to organize these functions into a coherent form. As a result, in the early 1970s various computer companies developed proprietary network architectures. A common feature to all of these was the grouping of the communication functions into related and manageable sets called **layers.** We saw in Chapter 1 that communication functions can be grouped according to the following tasks:

- The transport across a network of data from a process in one machine to the process at another machine.

[1]Definitions are from *The American Heritage Dictionary of the English Language,* Houghton Mifflin Co., 1978.

- The routing and forwarding of packets across multiple hops in a network.
- The transfer of a frame of data from one physical interface to another.

These layers of functions build on top of each other to enable communications. We use the term **network architecture** to refer to a set of protocols that specify how every layer is to function.

The decomposition of the overall communications problem into a set of layers is a first step to simplifying the design of the overall network. In addition the interaction between layers needs to be defined precisely. This is done through the *definition of the service* provided by each layer to the layer above, and through the definition of the *interface* between layers through which a service is requested and through which results are conveyed. A clearly defined service and interface allows a layer to invoke a service from the layer below without regard to how the service is implemented by any of the layers below. As long as the service is provided as specified, the implementation of the underlying layers can be changed. Also, new services that build on existing services can be introduced at any time, and in turn enable other new services at layers above. This provides flexibility in modifying and evolving the network. In contrast, a monolithic network design that uses a single large body of hardware and software to meet all the network requirements can quickly become obsolete and also is extremely difficult and expensive to modify. The layered approach accommodates incremental changes much more readily.

In this chapter we develop the notion of a layered architecture, and we provide examples from TCP/IP, the most important current network architecture. The discussion is organized as follows:

1. Web-browsing and e-mail applications are used to demonstrate the operation of a protocol within a layer and how it makes use of the communication services of the layer below. We introduce the HTTP, DNS, and SMTP application layer protocols in these examples.
2. The Open Systems Interconnection (OSI) reference model is discussed to show how the overall communication process can be organized into functions that are carried out in seven layers.
3. The TCP/IP architecture is introduced and compared to the OSI reference model. We present a detailed end-to-end example in a typical TCP/IP Internet. We use a network protocol analyzer to show the exchange of messages and packets in real networks. This section is key to seeing the big picture because it shows how all the layers work together.

Two optional sections present material that is useful in developing lab exercises and experiments involving TCP/IP:

4. We introduce Berkeley sockets, which allow the student to write applications that use the services provided by the TCP/IP protocols. We develop example programs that show the use of UDP and TCP sockets.
5. We introduce several important TCP/IP application layer protocols: Telnet, FTP, and HTTP. We also introduce several utilities and a network protocol analyzer that can be used as tools to study the operation of the Internet.

2.1 EXAMPLES OF PROTOCOLS, SERVICES, AND LAYERING

A **protocol** is a set of rules that governs how two or more communicating parties are to interact. When dealing with networks we run into a multiplicity of protocols, such as HTTP, FTP, and TCP. The purpose of a protocol is to provide some type of communication service. For example, the HTTP protocol enables the retrieval of web pages, and the TCP protocol enables the reliable transfer of streams of information between computers. In this chapter, we will see that the overall communications process can be organized into a stack of layers. Each layer carries out a specific set of communication functions using its own protocol, and each layer builds on the services of the layer below it.

This section uses concrete examples to illustrate what is meant by a protocol and to show how two adjacent layers interact. Together the examples also show the advantages of layering. The examples use two familiar applications, namely, e-mail and Web browsing. We present a simplified discussion of the associated protocols. Our purpose here is to relate familiar applications to the underlying network services that are the focus of this textbook.

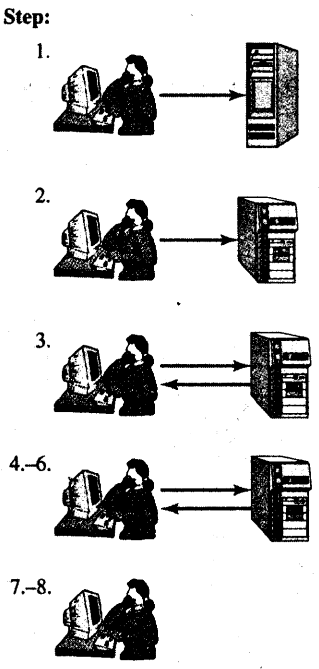
2.1.1 HTTP, DNS, and SMTP

All the examples discussed in this section involve a **client/server** application. A server process in a computer waits for incoming requests by listening to a **port.** A port is an address that identifies which process is to receive a message that is delivered to a given machine. Widely used applications have **well-known port numbers** assigned to their servers, so that client processes in other computers can readily make *requests* as required. The servers provide *responses* to those requests. The server software usually runs in the background and is referred to as a **daemon.** For example, httpd refers to the server daemon for HTTP.

EXAMPLE HTTP and Web Browsing

Let us consider an example of browsing through the World Wide Web (WWW). The WWW consists of a framework for accessing documents that are located in computers connected to the Internet. These documents are prepared using the *HyperText Markup Language (HTML)* and may consist of text, graphics, and other media and are interconnected by links that appear within the documents. The WWW is accessed through a browser program that displays the documents and allows the user to access other documents by clicking one of these links. Each link provides the browser with a uniform resource locator (URL) that specifies the name of the machine where the document is located as well as the name of the file that contains the requested document.

The *HyperText Transfer Protocol (HTTP)* specifies rules by which the client and server interact so as to retrieve a document. The rules also specify how the request and response are phrased. The protocol assumes that the client and server can exchange


Step:

1. The user clicks on a link to indicate which document is to be retrieved. The browser must determine the Internet address of the machine that contains the document. To do so, the browser sends a query to its local name server.

2. Once the address is known, the browser establishes a connection to the server process in the specified machine, usually a TCP connection. For the connection to be successful, the specified machine must be ready to accept TCP connections.

3. The browser runs a client version of HTTP, which issues a request specifying both the name of the document and the possible document formats it can handle.

4.–6. The machine that contains the requested document runs a server version of HTTP. It reacts to the HTTP request by sending an HTTP response which contains the desired document in the appropriate format.

7.–8. The user may start to view the document. The TCP connection is closed after a certain timeout period.

FIGURE 2.1 Retrieving a document from the web.

messages directly. In general, the client software needs to set up a two-way connection prior to the HTTP request.

Figure 2.1 and Table 2.1 show the sequence of events and messages that are involved in retrieving a document. In step 1 a user selects a document by clicking on its corresponding link. For example, the browser may extract the URL associated with the following link:

http://www.comm.utoronto.ca/comm.html

The client software must usually carry out a Domain Name System (DNS) query to determine the IP address corresponding to the host name, www.comm.utoronto.ca. (We discuss how this query is done in the next example.) The client software then sets up a TCP connection with the WWW server (the default is port 80) at the given IP address (step 2). The client end identifies itself by an **ephemeral port number** that is used only for the duration of the connection. The TCP protocol provides a reliable byte-stream transfer service that can be used to transmit files across the Internet.

After the connection is established, the client uses HTTP to request a document (step 3). The request message specifies the method or command (GET), the document (comm.html), and the protocol version that the browser is using (HTTP/1.1). The server daemon identifies the three components of the message and attempts to locate the file (step 4).

TABLE 2.1 Retrieving a document from the web: HTTP message exchange.

Event	Message Content
1. User selects document.	
2. Network software of client locates the server host and establishes a two-way connection.	
3. HTTP client sends message requesting document.	GET /comm.html HTTP/1.1
4. HTTP daemon listening on TCP port 80 interprets message.	
5. HTTP daemon sends a result code and a description of the information that the client will receive.	HTTP/1.1 200 OK
	Date: Mon, 06 Jan 2003 23:56:44 GMT
	Server: Apache/1.3.23 (Unix)
	Last Modified: 03 Sep 2002 02:58:36 GMT
	Content-Length: 8218
	Content-Type: text/html
6. HTTP daemon reads the file and sends requested file through the TCP port.	<html>
	<head><title></title>...
	<font face="Arial" >What is Communications?</font>
7. Text is displayed by client browser, which interprets the HTML format.	
8. HTTP daemon disconnects the connection after the connection is idle for some timeout period.	

In step 5 the daemon sends a status line and a description of the information that it will send. Result code 200 indicates that the client request was successful and that the document is to follow. The message also contains information about the server software, the length of the document (8218 bytes), and the content type of the document (text/html). If the request was for an image, the type might be image/gif. If the request is not successful, the server sends a different result code, which usually indicates the type of failure, for example, 404 when a document is not found.

In step 6 the HTTP daemon sends the file over the TCP connection. In the meantime, the client receives the file and displays it (step 7). The server maintains the TCP connection open so it can accept additional requests from the client. The server closes the TCP connection if it remains idle for some timeout period (step 8).

The HTTP example clearly indicates that a protocol is solely concerned with the interaction between the two peer processes, that is, the client and the server. The protocol assumes that the message exchange between peer processes occurs directly as shown in Figure 2.2. Because the client and server machines are not usually connected directly, a connection needs to be set up between them. In the case of HTTP, we require a two-way connection that transfers a stream of bytes in correct sequential order and without errors. The TCP protocol provides this type of *communication service* between two processes in two machines connected to a network. Each HTTP process inserts its

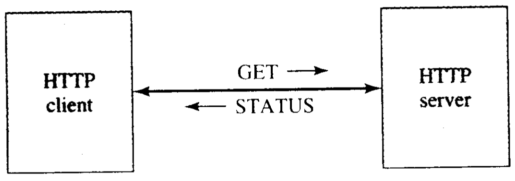


FIGURE 2.2 HTTP client/server interaction.

messages into a buffer, and TCP transmits the contents of the buffer to the other TCP in blocks of information called segments, as shown in Figure 2.3. Each segment contains port number information in addition to the HTTP message information. *HTTP is said to use the service provided by TCP in the layer below.* Thus the transfer of messages between HTTP client and server in fact is *virtual* and occurs *indirectly* via the TCP connection as shown in Figure 2.3. Later you will see that TCP, in turn, uses the service provided by IP.

It is worth noting exactly how the HTTP application protocol invokes the service provided by TCP. When the HTTP client software first needs to set up the TCP connection, the client does so by making a series of *socket system calls.* These calls are similar to function calls except that control is passed to the operating system kernel when a socket system call is made. A socket system call specifies a certain action and may contain parameters such as socket type, for example, TCP or UDP, and address information. Thus the interaction between the HTTP layer and the TCP layer takes place through these socket system calls.[2]

EXAMPLE DNS Query

The HTTP example notes that the client first needs to perform a DNS query to obtain the IP address corresponding to the domain name. This step is done by sending a

[2]Sockets are explained in detail in Section 2.4.

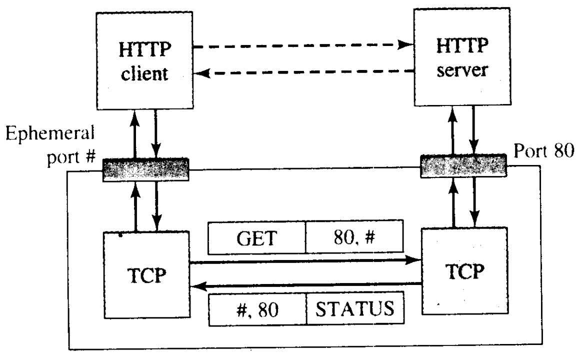


FIGURE 2.3 TCP provides a pipe between the HTTP client and HTTP server.

message to a DNS server. The **Domain Name System (DNS)** is a distributed database that resides in multiple machines on the Internet and is used to convert between names and addresses and to provide e-mail routing information. Each DNS machine maintains its own database and acts as a DNS server that other machines can query. Typically the requesting machine accesses a local name server, which, for example, may reside in a university department or at an ISP. These local name servers are able to resolve frequently used domain names into the corresponding IP addresses by caching recent information. When unable to resolve a name, the local name server may sometimes send a query to a root name server, of which there are currently 13 distributed globally. When a root server is unable to determine an IP address, it sends a query to an authoritative name server. Every machine on the Internet is required to register with at least two authoritative name servers. If a given name server cannot resolve the domain name, the queried name server will refer to another name server, and this process continues until a name server that can resolve the domain name is found.

We now consider a simple case where the resolution takes place in the first server. Table 2.2 shows the basic steps required for this example. After receiving the address request, a process in the host, called the resolver, composes the short message shown in step 2. The OPCODE value in the DNS message header indicates that the message is a standard query. The question portion of the query contains the following information: QNAME identifies the domain name that is to be translated. The DNS server can handle a variety of queries, and the type is specified by QTYPE. In the example, QTYPE = A requests a translation of a name to an IP address. QCLASS requests an Internet address (some name servers handle non-IP addresses). In step 3 the resolver sends the message to the local server using the datagram communication service UDP.

TABLE 2.2 DNS query and response.

Event	Message content
1. Application requests name to address translation.	

2. Resolver composes query message.	`Header: OPCODE=SQUERY`
	`Question:`
	`QNAME=tesla.comm.toronto.edu.,`
	`QCLASS=IN, QTYPE=A`

3. Resolver sends UDP datagram encapsulating the query message.	

4. DNS server looks up address and prepares response.	`Header: OPCODE=SQUERY,`
	`RESPONSE, AA`
	`Question: QNAME=`
	`tesla.comm.toronto.edu.,`
	`QCLASS=IN, QTYPE=A`
	`Answer: tesla.comm.toronto.edu.`
	`86400 IN A 128.100.11.1`

5. DNS sends UDP datagram encapsulating the response message.	

The short message returned by the server in step 4 has the Response and Authoritative Answer bits set in the header. This setting indicates that the response comes from an authority that manages the domain name. The question portion is identical to that of the query. The answer portion contains the domain name for which the address is provided. This portion is followed by the Time-to-Live field, which specifies the time in units of seconds that this information is to be cached by the client. Next are the two values for QCLASS and QTYPE. IN again indicates that it is an Internet address. Finally, the IP address of the domain name is given (128.100.11.1).

In this example the DNS query and response messages are transmitted by using the communication service provided by the **User Datagram Protocol (UDP).** The UDP client attaches a header to the user information to provide port information (port 53 for DNS) and encapsulates the resulting block in an IP packet. The UDP service is connectionless; no connection setup is required, and the datagram can be sent immediately. Because DNS queries and responses consist of short messages, UDP is ideally suited for conveying them.

The DNS example shows again how a protocol, in this case the DNS query protocol, is solely concerned with the interaction between the client and server processes. The example also shows how the transfer of messages between client and server, in fact, is virtual and occurs indirectly via UDP datagrams.

EXAMPLE SMTP and E-mail

Finally, we consider an e-mail example, using the **Simple Mail Transfer Protocol (SMTP).** Here a mail client application interacts with a local SMTP server to initiate the delivery of an e-mail message. The user prepares an e-mail message that includes the recipient's e-mail address, a subject line, and a body. When the user clicks Send, the mail application prepares a file with the above information and additional information specifying format, for example, plain ASCII or Multipurpose Internet Mail Extensions (MIME) to encode non-ASCII information. The mail application has the name of the local SMTP server and may issue a DNS query for the IP address. Table 2.3 shows the remaining steps involved in completing the transfer of the e-mail message to the local SMTP server.

Before the e-mail message can be transferred, the application process must set up a TCP connection to the local SMTP server (step 1). Thereafter, the SMTP protocol is used in a series of exchanges in which the client identifies itself, the sender of the e-mail, and the recipient (steps 2–8). The client then transfers the message that the SMTP server accepts for delivery (steps 9–12) and ends the mail session. The local SMTP server then repeats this process with the destination SMTP server. To locate the destination SMTP server, the local server may have to perform a DNS query of type MX (mail exchange). SMTP works best when the destination machine is always available. For this reason, users in a PC environment usually retrieve their e-mail from a mail server using the **Post Office Protocol version 3 (POP3)** instead.

TABLE 2.3 Sending e-mail.

Event	Message content
1. The mail application establishes a TCP connection (port 25) to its local SMTP server.	
2. SMTP daemon issues the following message to the client, indicating that it is ready to receive mail.	`220 tesla.comm.toronto.edu ESMTP Sendmail 8.9.0/8.9.0; Thu, 2 Jul 1998 05:07:59 -0400 (EDT)`
3. Client sends a HELO message and identifies itself.	`HELO bhaskara.comm.utoronto.ca`
4. SMTP daemon issues a 250 message, indicating the client may proceed.	`250 tesla.comm.toronto.edu Hello bhaskara.comm [128.100.10.9], pleased to meet you`
5. Client sends sender's address.	`MAIL FROM: <banerjea@comm.utoronto.ca>`
6. If successful, SMTP daemon replies with a 250 message.	`250 <banerjea@comm.utoronto.ca> ... Sender ok`
7. Client sends recipient's address.	`RCPT TO: <alg@nal.utoronto.ca>`
8. A 250 message is returned.	`250 <alg@nal.utoronto.ca> ... Recipient ok`
9. Client sends a DATA message requesting permission to send the mail message.	`DATA`
10. The daemon sends a message giving the client permission to send.	`354 Enter mail, end with "." on a line by itself`
11. Client sends the actual text.	`Hi Al, This section on email sure needs a lot of work...`
12. Daemon indicates that the message is accepted for delivery. A message ID is returned.	`250 FAA00803 Message accepted for delivery`
13. Client indicates that the mail session is over.	`QUIT`
14. Daemon confirms the end of the session.	`221 tesla.comm.toronto.edu closing connection`

2.1.2 TCP and UDP Transport Layer Services

The e-mail, DNS query, and HTTP examples show how multiple protocols can operate by using the communication services provided by the TCP and UDP protocols. Both the TCP and UDP protocols operate by using the connectionless packet network service provided by IP.

UDP provides connectionless transfer of datagrams between processes in hosts attached to the Internet. UDP provides port numbering to identify the source and destination processes in each host. UDP is simple and fast but provides no guarantees in terms of delivery or sequence addressing.

TCP provides for reliable transfer of a byte stream between processes in hosts attached to the Internet. The processes write bytes into a buffer for transfer across the Internet by TCP. TCP is considerably more complex than UDP. TCP involves the establishment of a connection between the two processes. To provide their service, the TCP entities implement error detection and retransmission as well as flow control algorithms (discussed in Chapters 5 and 8). In addition, TCP also implements congestion control, which regulates the flow of segments into the network. This topic is discussed in Chapters 7 and 8.

Indeed, an entire suite of protocols has been developed to operate on top of TCP and UDP, thereby demonstrating the usefulness of the layering concept. New services can be quickly developed by building on the services provided by existing layer protocols.

PEER-TO-PEER FILE SHARING

File-sharing applications such as Napster and Gnutella became extremely popular as a means of exchanging MP3 audio and other files. The essence of these peer-to-peer applications is that ordinary PCs ("peers") attached to the Internet can act not only as clients, but also as transient file servers while the applications are activated. When a peer is interested in finding a certain file, it sends a query. The response provides a list of peers that have the file and additional information such as the speed of each peer's connection to the Internet. The requesting peer can then set up a TCP connection to one of the peers in the list and proceed to retrieve the file.

The technically difficult part in peer-to-peer file sharing is maintaining the database of peers that are connected at a given point in time and the files that they have available for sharing. The Napster approach used a centralized database that peers could contact when they became available for file sharing and/or when they needed to make a query. The Gnutella approach uses a distributed approach where the peers organize themselves into an overlay network by keeping track of peers that are assigned to be adjacent to them. A query from a given peer is then broadcast by sending the query to each neighbor, their neighbors' neighbors, and so on up to some maximum number of hops.

Peer-to-peer file sharing provides another example of how new services and applications can be deployed very quickly over the Internet. Peer-to-peer file sharing also brings up many legal, commercial, and cultural issues that will require many years to resolve.

2.2 THE OSI REFERENCE MODEL

The early network architectures developed by various computer vendors were not compatible with each other. This situation had the effect of locking in customers with a single vendor. As a result, there was pressure in the 1970s for an open systems architecture that would eventually lead to the design of computer network equipment that could communicate with each other. This desire led to an effort in the International Organization for Standardization (ISO) first to develop a reference model for open systems interconnection (OSI) and later to develop associated standard protocols. *The OSI reference model partitioned the communications process into seven layers and* *provided a framework for talking about the overall communications process* and hence was intended to facilitate the development of standards. The OSI work also provided a unified view of layers, protocols and services. This unified view has provided the basis for the development of networking standards to the present day.

2.2.1 The Seven-Layer OSI Reference Model

Consider an application that involves communications between a process in computer A and a process in computer B. The **OSI reference model** divides the basic communication functions required for computers A and B to communicate into the seven layers shown in Figure 2.4. In this section, we will discuss the functions of the seven layers starting from the bottom (physical layer) to the top (application layer). The reader should compare the definition of the OSI layers to the elements described for the telegraph, telephone, and computer network architectures discussed in Chapter 1.

The **physical layer** deals with the transfer of *bits* over a communication channel, for example, the digital transmission system and the transmission media such as copper wire pairs, coaxial cable, radio, or optical fiber. The layer is concerned with the particular choice of system parameters such as voltage levels and signal durations. The layer is also concerned with the procedures to set up and release the physical connection, as well as with mechanical aspects such as socket type and number of pins. For example, an Ethernet physical layer standard defines the connector and signal interfaces in a PC.

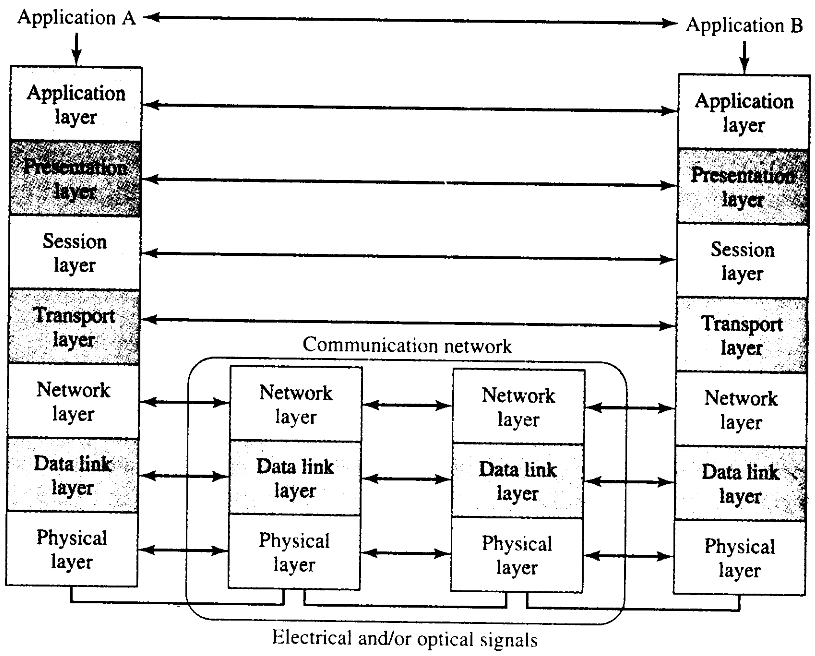


FIGURE 2.4 The seven-layer OSI reference model.

The **data link layer** provides for the transfer of **frames** (blocks of information) across a transmission link that *directly* connects two nodes. The data link layer inserts *framing* information in the sequence of transmitted bits to indicate the boundaries of the frames. It also inserts control and address information in the header and check bits to enable recovery from transmission errors, as well as flow control. The data link control is particularly important when the transmission link is prone to transmission errors. Historically, the data link layer has included the case where multiple terminals are connected to a host computer in point-to-multipoint fashion. In Chapter 5 we will discuss High-level Data Link Control (HDLC) protocol and Point-to-Point Protocol (PPP), which are two standard data link controls that are in wide use.

The OSI data link layer was defined so that it included the functions of *LANs*, which are characterized by the use of broadcast transmissions. The notion of a "link," then, includes the case where multiple nodes are connected to a broadcast medium. As before, frames flow directly between nodes. A *medium access control* procedure is required to coordinate the transmissions from the machines into the medium. A flat addressing space is used to enable machines to listen and recognize frames that are destined to them. Later in this chapter we will discuss the Ethernet LAN standard.

The **network layer** provides for the transfer of data in the form of **packets** across a communication *network*. One key aspect of the network layer is the use of a hierarchical addressing scheme that identifies the point of attachment to the network and that can accommodate a large number of network users. A key aspect of the packet transfer service is the routing of the packets from the source machine to the destination machine, typically traversing a number of transmission links and network nodes where routing is carried out. By *routing protocol* we mean the procedure that is used to select paths across a network. The nodes in the network must work together to perform the routing effectively. This function makes the network layer the most complex in the reference model. The network layer is also responsible for dealing with the *congestion* that occurs from time to time due to temporary surges in packet traffic.

When the two machines are connected to the same packet-switching network as in Figure 2.5, a single address space and routing procedure are used. However, when the two machines are connected to different networks, the transfer of data must traverse two or more networks that possibly differ in their internal routing and addressing

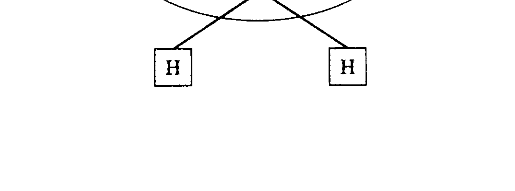


FIGURE 2.5 A packet-switching network using a uniform routing procedure.

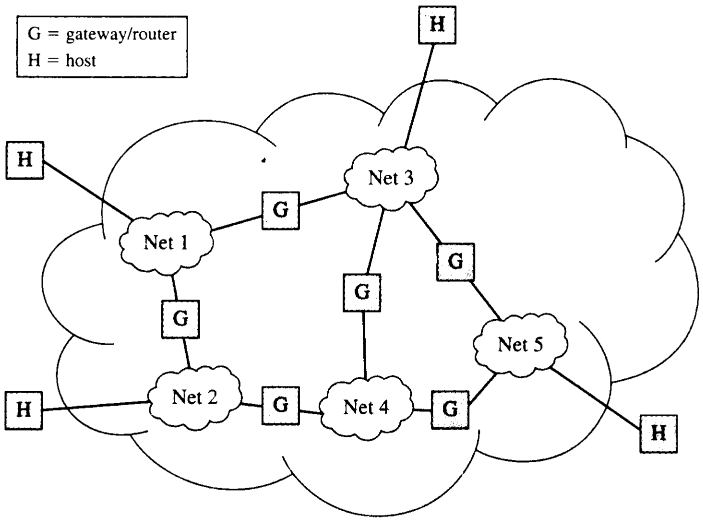


FIGURE 2.6 An internetwork.

scheme. In this case **internetworking** protocols are necessary to route the data between gateways/routers that connect the intermediate networks, as shown in Figure 2.6. The internetworking protocols must also deal with differences in addressing and differences in the size of the packets that are handled within each network. This *internet sublayer* of the network layer assumes the responsibility for hiding the details of the underlying network(s) from the upper layers. This function is particularly important given the large and increasing number of available network technologies for accomplishing packet transfer.

As shown in Figure 2.4, each intermediate node in the network must implement the lower three layers. Thus one pair of network layer entities exists for each hop of the path required through the network. Note that the network layer entities in the source and destination machines are *not* peer processes, that is, if there are intermediate nodes between them, they do not talk directly to each other.

The **transport layer** is responsible for the end-to-end transfer of messages from a process in the source machine to a process in the destination machine. The transport layer protocol accepts messages from its higher layers and prepares blocks of information called **segments** or datagrams for transfer between end machines. The transport layer uses the services offered by the underlying network or internetwork to provide the session layer with a transfer of messages that meets a certain quality of service. The transport layer can provide a variety of services. At one extreme the transport layer may provide a connection-oriented service that involves the error-free transfer of a sequence of bytes or messages. The associated protocol carries out error detection and recovery, and sequence and flow control. At the other extreme the transport layer may instead provide an unconfirmed connectionless service that involves the transfer of individual messages. In this case the role of the transport layer is to provide the appropriate address information so that the messages can be delivered to the appropriate destination process. The transport layer may be called upon to segment messages that are too long into shorter blocks of information for transfer across a network and to reassemble these messages at the destination.

In TCP/IP networks, processes typically access the transport layer through **socket** interfaces that are identified by a port number. We discuss the socket interface in the Berkeley UNIX application programming interface (API) in an optional section later in this chapter.

The transport layer can be responsible for setting up and releasing connections across the network. To optimize the use of network services, the transport layer may multiplex several transport layer connections onto a single network layer connection. On the other hand, to meet the requirements of a high throughput transport layer connection, the transport layer may use splitting to support its connection over several network layer connections.

Note from Figure 2.4 that the top four layers are end to end and involve the interaction of peer processes across the network. In contrast the lower two layers of the OSI reference model involve interaction of peer-to-peer processes across a single hop.

The **session layer** can be used to control the manner in which data are exchanged. For example, certain applications require a half-duplex dialog where the two parties take turns transmitting information. Other applications require the introduction of synchronization points that can be used to mark the progress of an interaction and can serve as points from which error recovery can be initiated. For example, this type of service may be useful in the transfer of very long files over connections that have short times between failures.

The **presentation layer** is intended to provide the application layer with independence from differences in the representation of data. In principle, the presentation layer should first convert the machine-dependent information provided by application A into a machine-independent form, and later convert the machine-independent form into a machine-dependent form suitable for application B. For example, different computers use different codes for representing characters and integers, and also different conventions as to whether the first or last bit is the most significant bit.

Finally, the purpose of the **application layer** is to provide services that are frequently required by applications that involve communications. In the WWW example the browser application uses the HTTP application-layer protocol to access a WWW document. Application layer protocols have been developed for file transfer, virtual terminal (remote log-in), electronic mail, name service, network management, and other applications.

In general each layer adds a header, and possibly a trailer, to the block of information it accepts from the layer above. Figure 2.7 shows the headers and trailers that are added as a block of application data works its way down the seven layers. At the destination each layer reads its corresponding header to determine what action to take and it eventually passes the block of information to the layer above after removing the header and trailer.

In addition to defining a reference model, an objective of the ISO activity was the development of *standards* for computer networks. This objective entailed *specifying the particular protocols* that were to be used in various layers of the OSI reference model. However, in the time that it took to develop the OSI protocol standards, the *TCP/IP network architecture* emerged as an alternative for open systems interconnection. The free distribution of TCP/IP as part of the Berkeley UNIX® ensured the development of

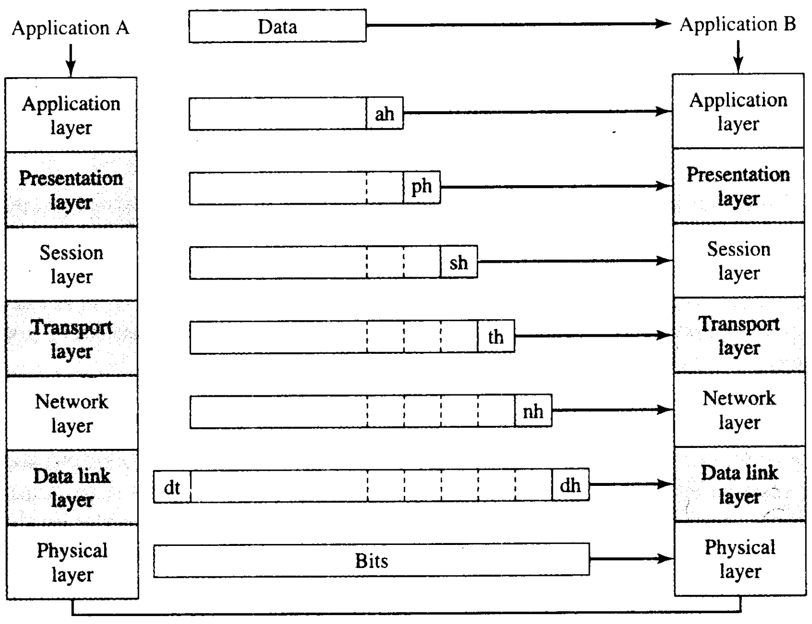


FIGURE 2.7 Headers and trailers are added to a block of data as it moves down the layers.

numerous applications at various academic institutions and the emergence of a market for networking software. This situation eventually led to the development of the global Internet and to the dominance of the TCP/IP network architecture.

2.2.2 Unified View of Layers, Protocols, and Services

A lasting contribution from the development of the OSI reference model was the development of a unified view of layers, protocols, and services. Similar requirements occur at different layers in a network architecture, for example, in terms of addressing, multiplexing, and error and flow control. This unified view enables a common understanding of the protocols that are found in different layers. In each layer a process on one machine carries out a conversation with a **peer process** on the other machine across a *peer interface,* as shown in Figure 2.8.[3] In OSI terminology the processes at layer n are referred to as **layer n entities.** Layer n entities communicate by exchanging **protocol data units (PDUs).** Each PDU contains a **header,** which contains protocol control information, and usually user information. The behavior of the layer n entities is governed by a set of rules or conventions called the **layer n protocol.** In the HTTP example the HTTP client and server applications acted as peer processes. The processes that carry

[3]Peer-to-peer protocols are present in every layer of a network architecture. In Chapter 5 we present detailed examples of peer-to-peer protocols.

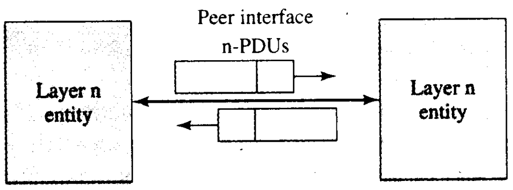


FIGURE 2.8 Peer-to-peer communication.

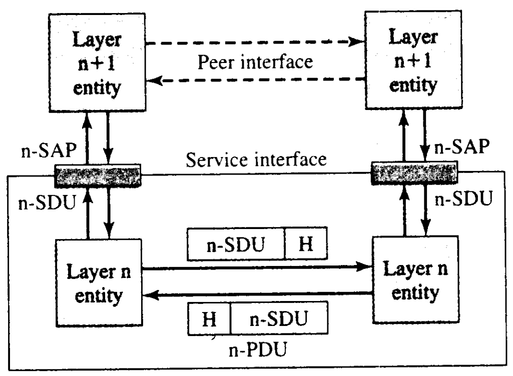


FIGURE 2.9 Layer services: SDUs are exchanged between layers while PDUs are exchanged within a layer.

out the transmitter and receiver functions of TCP also constitute peer processes at the layer below.

The communication between peer processes is usually virtual in the sense that no direct communication link exists between them. *For communication to take place, the layer n + 1 entities make use of the services provided by layer n.* The transmission of the layer n + 1 PDU is accomplished by passing a block of information from layer n + 1 to layer n through a software port called the layer n **service access point (SAP)** across a *service interface,* as shown in Figure 2.9. Each SAP is identified by a unique identifier (for example, recall that a WWW server process passes information to a TCP process through a Transport-SAP or port number 80). The block of information passed between layer n and layer n + 1 entities consists of control information and a layer n **service data unit (SDU),** which is the layer n + 1 PDU itself. The layer n entity uses the control information to form a header that is attached to the SDU to produce the layer n PDU. Upon receiving the layer n PDU, the layer n peer process uses the header to execute the layer n protocol and, if appropriate, to deliver the SDU to the corresponding layer n + 1 entity. The communication process is completed when the SDU (layer n + 1 PDU) is passed to the layer n + 1 peer process.[4]

In principle, the layer n protocol does not interpret or make use of the information contained in the SDU.[5] We say that the layer n SDU, which is the layer n + 1 PDU, is *encapsulated* in the layer n PDU. This process of **encapsulation** narrows the scope of the dependencies between adjacent layers to the service definition only. In other

[4]It may be instructive to reread this paragraph where a DNS query message constitutes the layer n + 1 PDU and a UDP datagram constitutes the layer n PDU.

[5]On the other hand, accessing some of the information "hidden" inside the SDU can sometimes be useful.

words, *layer n + 1, as a user of the service provided by layer n, is only interested in the correct execution of the service required to transfer its PDUs. The details of the implementation of the layers below layer n + 1 are irrelevant.*

The service provided by layer n typically involves accepting a block of information from layer n + 1, transferring the information to its peer process, which in turn delivers the block to the user at layer n + 1. The service provided by a layer can be connection oriented or connectionless. A **connection-oriented service** has three phases.

1. Establishing a connection between two layer n SAPs. The setup involves negotiating connection parameters as well as initializing "state information" such as the sequence numbers, flow control variables, and buffer allocations.
2. Transferring n-SDUs using the layer n protocol.
3. Tearing down the connection and releasing the various resources allocated to the connection.

In the HTTP example in Section 2.1, the HTTP client process uses the connection services provided by TCP to transfer the HTTP PDU, which consists of the request message. A TCP connection is set up between the HTTP client and server processes, and the TCP transmitter/receiver entities carry out the TCP protocol to provide a reliable message stream service for the exchange of HTTP PDUs. The TCP connection is later released after one or more HTTP responses have been received.

Connectionless service does not require a connection setup, and each SDU is transmitted directly through the SAP. In this case the control information that is passed from layer n + 1 to layer n must contain all the address information required to transfer the SDU. In the DNS example in Section 2.1, UDP provides a connectionless service for the exchange of DNS PDUs. No connection is established between the DNS client and server processes.

In general, it is not necessary for the layers to operate in the same connection mode. Thus for example, TCP provides a connection-oriented service but builds on the connectionless service provided by IP.

The services provided by a layer can be **confirmed** or **unconfirmed** depending on whether the sender must eventually be informed of the outcome. For example, connection setup is usually a confirmed service. Note that a connectionless service can be confirmed or unconfirmed depending on whether the sending entity needs to receive an acknowledgment.

Information exchanged between entities can range from a few bytes to multi-megabyte blocks or continuous byte streams. Many transmission systems impose a limit on the maximum number of bytes that can be transmitted as a unit. For example, Ethernet LANs have a maximum transmission size of approximately 1500 bytes. Consequently, when the number of bytes that needs to be transmitted exceeds the maximum transmission size of a given layer, it is necessary to divide the bytes into appropriate-sized blocks.

In Figure 2.10a a layer n SDU is too large to be handled by the layer n − 1, and so **segmentation** and **reassembly** are applied. The layer n SDU is *segmented* into multiple layer n PDUs that are then transmitted using the services of layer n − 1. The layer n entity at the other side must *reassemble* the original layer n SDU from the sequence of layer n PDUs it receives.

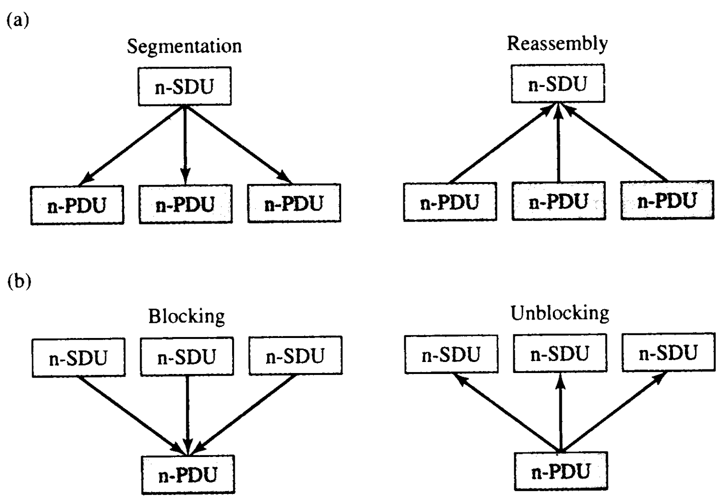


FIGURE 2.10 Segmentation/reassembly and blocking/unblocking.

On the other hand, it is also possible that the layer n SDUs are so small as to result in inefficient use of the layer n − 1 services, and so **blocking** and **unblocking** may be applied. In this case, the layer n entity may *block* several layer n SDUs into a single layer n PDU as shown in Figure 2.10b. The layer n entity on the other side must then *unblock* the received PDU into the individual SDUs.

Multiplexing involves the sharing of a layer n service by *multiple* layer n + 1 users. Figure 2.11 shows the case where each layer n + 1 user passes its SDUs for transfer using the service of a single layer n entity. **Demultiplexing** is carried out by the layer n entity at the other end. When the layer n PDUs arrive at the other end of the connection, the SDUs are recovered and must then be delivered to the appropriate layer n + 1 user. Clearly a *multiplexing tag* is needed in each PDU to determine which user an SDU belongs to. As an example consider the case where several application layer processes share the

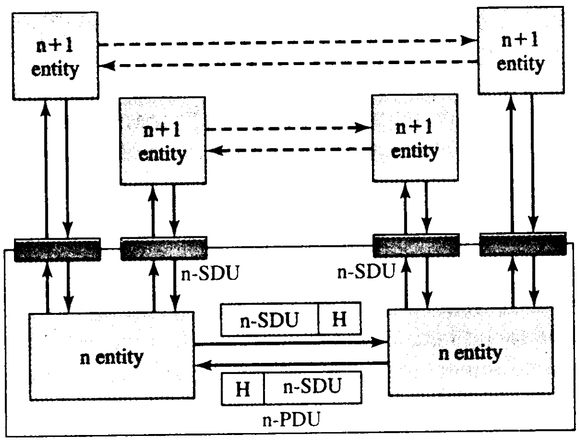


FIGURE 2.11 Multiplexing involves sharing of layer n service by multiple layer n + 1 users.

datagram services of UDP. Each application layer process passes its SDU through its socket to the UDP entity. UDP prepares a datagram that includes the source port number, the destination port number, as well as the IP address of the source and destination machines. The server-side port number is a *well-known port number* that unambiguously identifies the process that is to receive the SDU at the server end. The client-side port number is an ephemeral number that is selected when the socket for the application is established. Demultiplexing can then be carried out unambiguously at each UDP entity by directing an arriving SDU to the port number indicated in the datagram.

Splitting involves the use of several layer n services to support a single layer n + 1 user. The SDUs from the single user are directed to one of several layer n entities, which in turn transfer the given SDU to a peer entity at the destination end. **Recombining** takes place at the destination where the SDUs recovered from each of the layer n entities are passed to the layer n + 1 user. Sequence numbers may be required to reorder the received SDUs.

Multiplexing is used to achieve more efficient use of communications services. Multiplexing is also necessary when only a single connection is available between two points. Splitting can be used to increase reliability in situations where the underlying transfer mechanism is unreliable. Splitting is also useful when the transfer rate required by the user is greater than the transfer rate available from individual services.

In closing we re-iterate: Similar needs occur at different layers and these can be met by a common set of services such as those introduced here.

2.3 OVERVIEW OF TCP/IP ARCHITECTURE

The TCP/IP network architecture is a set of protocols that allows communication across multiple diverse networks. The architecture evolved out of research that had the original objective of transferring packets across three different packet networks: the ARPANET packet-switching network, a packet radio network, and a packet satellite network. The military orientation of the research placed a premium on robustness with regard to failures in the network and on flexibility in operating over diverse networks. This environment led to a set of protocols that are highly effective in enabling communications among the many different types of computer systems and networks. Indeed, the Internet has become the primary fabric for interconnecting the world's computers. In this section we introduce the TCP/IP network architecture. The details of specific protocols that constitute the TCP/IP network architecture are discussed in later chapters.

2.3.1 TCP/IP Architecture

Figure 2.12a shows the **TCP/IP network architecture,** which consists of four layers. The application layer provides services that can be used by other applications. For example, protocols have been developed for remote login, for e-mail, for file transfer, and for network management. The TCP/IP application layer incorporates the functions of the

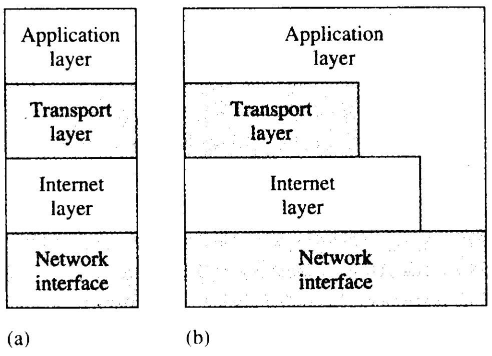


FIGURE 2.12 TCP/IP network architecture.

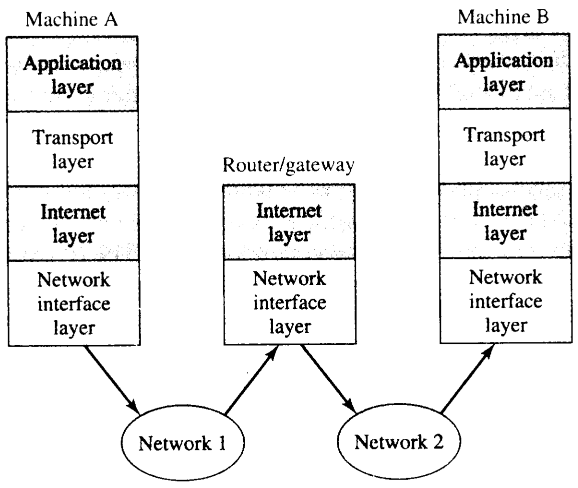


FIGURE 2.13 The internet layer and network interface layers.

top three OSI layers. The HTTP protocol discussed in Section 2.1 is actually a TCP/IP application layer protocol. Recall that the HTTP request message included format information and the HTTP protocol defined the dialogue between the client and server.

The TCP/IP application layer programs are intended to run directly over the transport layer. Two basic types of services are offered in the transport layer. The first service consists of reliable connection-oriented transfer of a byte stream, which is provided by the *Transmission Control Protocol (TCP)*. The second service consists of best-effort connectionless transfer of individual messages, which is provided by the *User Datagram Protocol (UDP)*. This service provides no mechanisms for error recovery or flow control. UDP is used for applications that require quick but not necessarily reliable delivery.

The TCP/IP model does not require strict layering, as shown in Figure 2.12b. In other words, the application layer has the option of bypassing intermediate layers. For example, an application layer may run directly over the internet layer.

The **internet layer** handles the transfer of information across multiple networks through the use of gateways/routers, as shown in Figure 2.13. The internet layer corresponds to the part of the OSI network layer that is concerned with the transfer of packets between machines that are connected to different networks. It must therefore deal with the routing of packets from router to router across these networks. A key aspect of routing in the internet layer is the definition of *globally unique addresses* for machines that are attached to the Internet. The internet layer provides a single service, namely, *best-effort connectionless packet transfer.* IP packets are exchanged between routers without a connection setup; the packets are routed independently, and so they may traverse different paths. For this reason, IP packets are also called **datagrams.** The connectionless approach makes the system robust; that is, if failures occur in the network, the packets are routed around the points of failure; there is no need to set up the connections again. The gateways that interconnect the intermediate networks may discard packets when congestion occurs. The responsibility for recovery from these losses is passed on to the transport layer.

Finally, the **network interface layer** is concerned with the network-specific aspects of the transfer of packets. As such, it must deal with part of the OSI network layer and data link layer. Various interfaces are available for connecting end computer systems to specific networks such as ATM, frame relay, Ethernet, and token ring. These networks are described in later chapters.

The network interface layer is particularly concerned with the protocols that access the intermediate networks. At each gateway the network access protocol encapsulates the IP packet into a packet or frame of the underlying network or link. The IP packet is recovered at the exit gateway of the given network. This gateway must then encapsulate the IP packet into a packet or frame of the type of the next network or link. This approach provides a clear separation of the internet layer from the technology-dependent network interface layer. This approach also allows the internet layer to provide a data transfer service that is transparent in the sense of not depending on the details of the underlying networks. The next section provides a detailed example of how IP operates over the underlying networks.

Figure 2.14 shows some of the protocols of the TCP/IP protocol suite. The figure shows two of the many protocols that operate over TCP, namely, HTTP and SMTP.

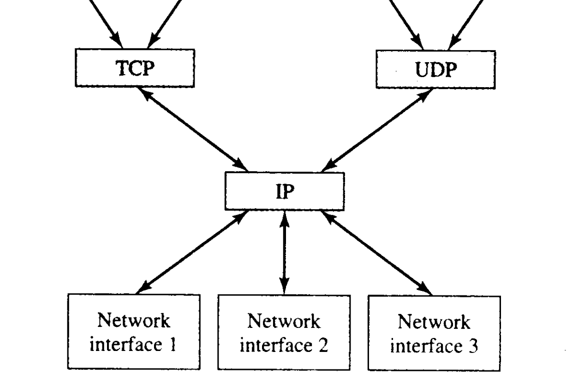


FIGURE 2.14 TCP/IP protocol graph.

The figure also shows DNS and Real-Time Protocol (RTP), which operate over UDP. The transport layer protocols TCP and UDP, on the other hand, operate over IP. Many network interfaces are defined to support IP. The salient part of Figure 2.14 is that all higher-layer protocols access the network interfaces through IP. This feature provides the capability to operate over multiple networks. The IP protocol is complemented by additional protocols (ICMP, IGMP, ARP, RARP) that are required to operate an internet. These protocols are discussed in Chapter 8.

The hourglass shape of the TCP/IP protocol graph underscores the features that make TCP/IP so powerful. The operation of the single IP protocol over various networks provides independence from the underlying network technologies. The communication services of TCP and UDP provide a network-independent platform on which applications can be developed. By allowing multiple network technologies to coexist, the Internet is able to provide ubiquitous connectivity and to achieve enormous economies of scale.

2.3.2 TCP/IP Protocol: How the Layers Work Together

We now provide a detailed example of how the layering concepts discussed in the previous sections are put into practice in a typical TCP/IP network scenario. We show

- Examples of each of the layers.
- How the layers interact across the interfaces between them.
- How the PDUs of a layer are built and what key information is in the header.
- The relationship between physical addresses and IP addresses.
- How an IP packet or datagram is routed across several networks.

We first consider a simplified example, and then we present an example showing PDUs captured in a live network by a network protocol analyzer. These examples will complete our goal of providing the big picture of networking. In the remainder of the book we systematically examine the details of the various components and aspects of networks.

Consider the network configuration shown in Figure 2.15a. A server, a workstation, and a router are connected to an Ethernet LAN, and a remote PC is connected to the router through a point-to-point link. From the point of view of IP, the Ethernet LAN and the point-to-point link constitute two different networks as shown in Figure 2.15b.

IP ADDRESSES AND PHYSICAL ADDRESSES

Each host in the Internet is identified by a **globally unique IP address.** Strictly speaking, the IP address identifies the host's network interface rather than the host itself. A node that is attached to two or more physical networks is called a router. In this example the router attaches to two networks with each network interface assigned to a unique IP address. An IP address is divided into two parts: a *network id* and a *host id*. The network id must be obtained from an organization authorized to issue IP addresses. In this example we use simplified notation and assume that the Ethernet has net id 1 and that the point-to-point link has a net id 2. In the Ethernet we suppose that the server has IP address (1,1), the workstation has IP address (1,2), and the router has address (1,3). In the point-to-point link, the PC has address (2,2), and the router has address (2,1).

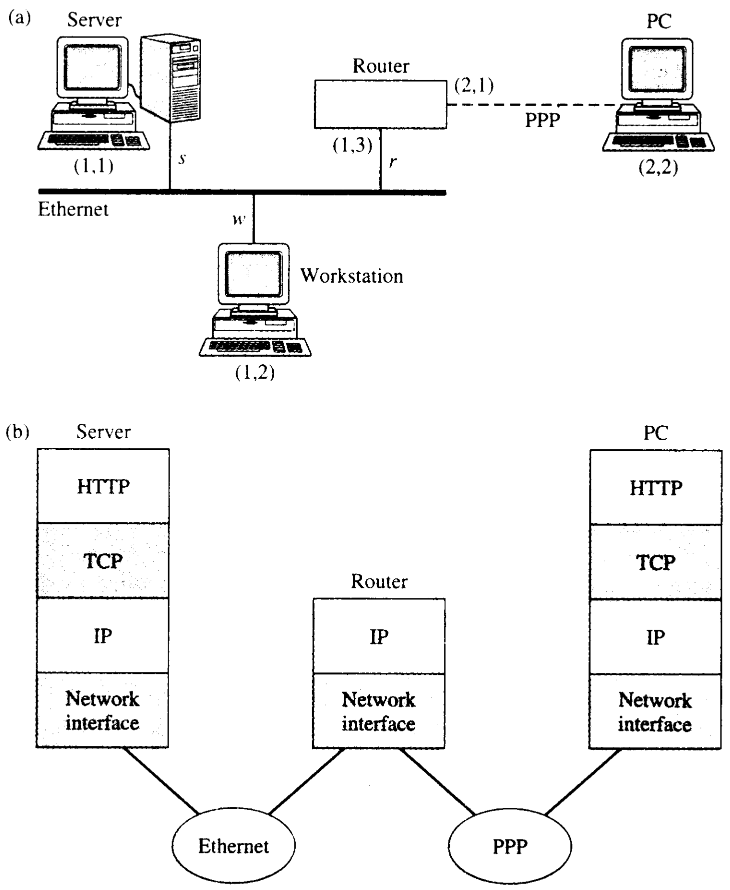


FIGURE 2.15 An example of an internet consisting of an Ethernet LAN and a point-to-point link: (a) physical configuration view and (b) IP network view.

On a LAN the attachment of a device to the network is often identified by a **physical address.** The format of the physical address depends on the particular type of network. For example, Ethernet LANs use 48-bit addresses. Each Ethernet network interface card (NIC) is issued a globally unique medium access control (MAC) or physical address. When a NIC is used to connect a machine to any Ethernet LAN, all machines in the LAN are automatically guaranteed to have unique addresses. Thus the router, server, and workstation also have physical addresses designated by *r*, *s*, and *w*, respectively.

SENDING AND RECEIVING IP DATAGRAMS

First, let us consider the case in which the workstation wants to send an IP datagram to the server. The IP datagram has the workstation's IP address and the server's IP address in the IP packet header. We suppose that the IP address of the server is known. The IP entity in the workstation looks at its routing table to see whether it has an entry for the

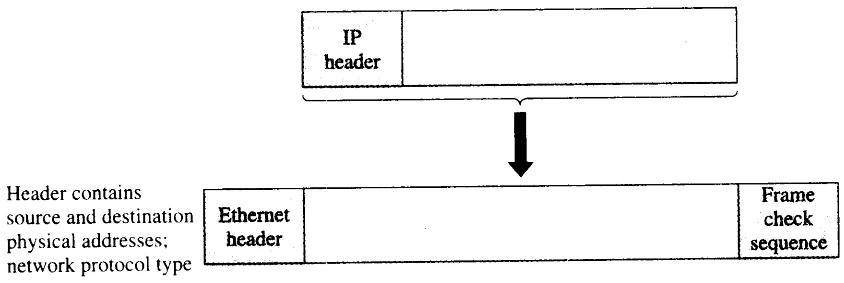

IP header
Header contains source and destination physical addresses; network protocol type
Ethernet header
Frame check sequence

FIGURE 2.16 IP datagram is encapsulated in an Ethernet frame.

complete IP address. It finds that the server is directly connected to the same network and that the server has physical address s.[6] The IP datagram is passed to the Ethernet device driver, which prepares an Ethernet frame as shown in Figure 2.16. The header in the frame contains the source physical address, w, and the destination physical address, s. The header also contains a protocol type field that is set to the value that corresponds to IP. The type field is required because the Ethernet may be carrying packets for other non-IP protocols. The Ethernet frame is then broadcast over the LAN. The server's NIC recognizes that the frame is intended for its host, so the card captures the frame and examines it. The NIC finds that the protocol type field is set to IP and therefore passes the IP datagram up to the IP entity.

Next let us consider the case in which the server wants to send an IP datagram to the personal computer. The PC is connected to the router through a point-to-point link that we assume is running PPP as the data link control.[7] We suppose that the server knows the IP address of the PC and that the IP addresses on either side of the link were negotiated when the link was set up. The IP entity in the server looks at its routing table to see whether it has an entry for the complete IP address of the PC. We suppose that it doesn't. The IP entity then checks to see whether it has a routing table entry that matches the network id portion of the IP portion of the IP address of the PC. Again we suppose that the IP entity does not find such an entry. The IP entity then checks to see whether it has an entry that specifies a default router that is to be used when no other entries are found. We suppose that such an entry exists and that it specifies the router with address (1,3).

The IP datagram is passed to the Ethernet device driver, which prepares an Ethernet frame. The header in the frame contains the source physical address, s, and the destination physical address, r. However, the IP datagram in the frame contains the destination IP address of the PC, (2,2), not the destination IP address of the router. The Ethernet frame is then broadcast over the LAN. The router's NIC captures the frame and examines it. The card passes the IP datagram up to its IP entity, which discovers that the IP datagram is not for itself but is to be routed on.

[6] If the IP entity does not know the physical address corresponding to the IP address of the server, the entity uses the Address Resolution Protocol (ARP) to find it. ARP is discussed in Chapter 8.

[7] PPP is discussed in Chapter 5.

The routing tables at the router show that the machine with address (2,2) is connected directly on the other side of the point-to-point link. The router encapsulates the IP datagram in a PPP frame that is similar to that of the Ethernet frame shown in Figure 2.16. However, the frame does not require physical address information, since there is only one "other side" of the link. The PPP receiver at the PC receives the frame, checks the protocol type field, and passes the IP datagram to its IP entity.

HOW THE LAYERS WORK TOGETHER

The preceding discussion shows how IP datagrams are sent across an internet. Next let's complete the picture by seeing how things work at the higher layers. Consider the browser application discussed in the beginning of the chapter. We suppose that the user at the PC has clicked on a web link of a document contained in the server and that a TCP connection has already been established between the PC and the server.[8] Consider what happens when the TCP connection is confirmed at the PC. The HTTP request message GET is passed to the TCP layer, which encapsulates the message into a TCP segment as shown in Figure 2.17. The TCP segment contains an ephemeral port number for the client process, say, c, and a well-known port number for the server process, 80 for HTTP.

The TCP segment is passed to the IP layer, which in turn encapsulates the segment into an Internet packet. The IP packet header contains the IP addresses of the sender, (2,2), and the destination, (1,1). The header also contains a protocol field, which designates the layer that is operating above IP, in this case TCP. The IP datagram is then encapsulated using PPP and sent to the router, which routes the datagram to the server using the procedures discussed above. Note that the router encapsulates the IP datagram for the server in an Ethernet frame.

Eventually the server NIC captures the Ethernet frame and extracts the IP datagram and passes it to the IP entity. The protocol field in the IP header indicates that a TCP segment is to be extracted and passed on to the TCP layer. The TCP layer, in turn, uses the port number to find out that the message is to be passed to the HTTP server process. A problem arises at this point: The server process is likely to be simultaneously handling multiple connections to multiple clients. All these connections have the same destination IP address; the same destination port number, 80; and the same protocol type, TCP. How does the server know which connection the message corresponds to? The answer is in how an end-to-end process-to-process connection is specified.

The source port number, the source IP address, and the protocol type are said to define the sender's **socket address.** Similarly, the destination port number, the destination IP address, and the protocol type define the destination's socket address. Together the source socket address and the destination socket address uniquely specify the connection between the HTTP client process and the HTTP server process. For example, in the earlier HTTP example the sender's socket is (TCP, (2,2), c), and the destination's socket is (TCP, (1,1), 80). The combination of these five parameters (TCP, (2,2), c, (1,1), 80) uniquely specify the process-to-process connection.

[8]The details of how a TCP connection is set up are described in Chapter 8.

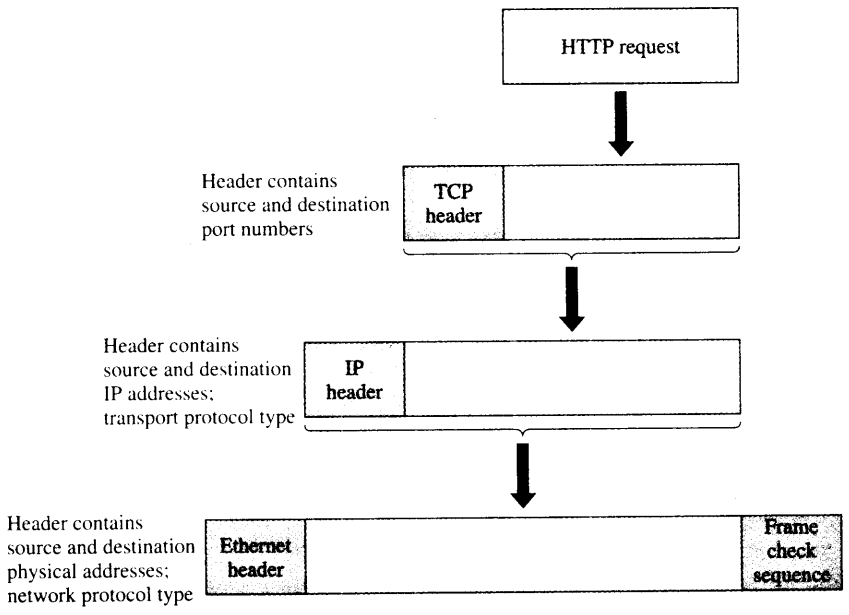

HTTP request
Header contains source and destination port numbers
TCP header
Header contains source and destination IP addresses; transport protocol type
IP header
Header contains source and destination physical addresses; network protocol type
Ethernet header
Frame check sequence

FIGURE 2.17 Encapsulation of PDUs in TCP/IP and addressing information in the headers. (Ethernet header is replaced with PPP header on a PPP link.)

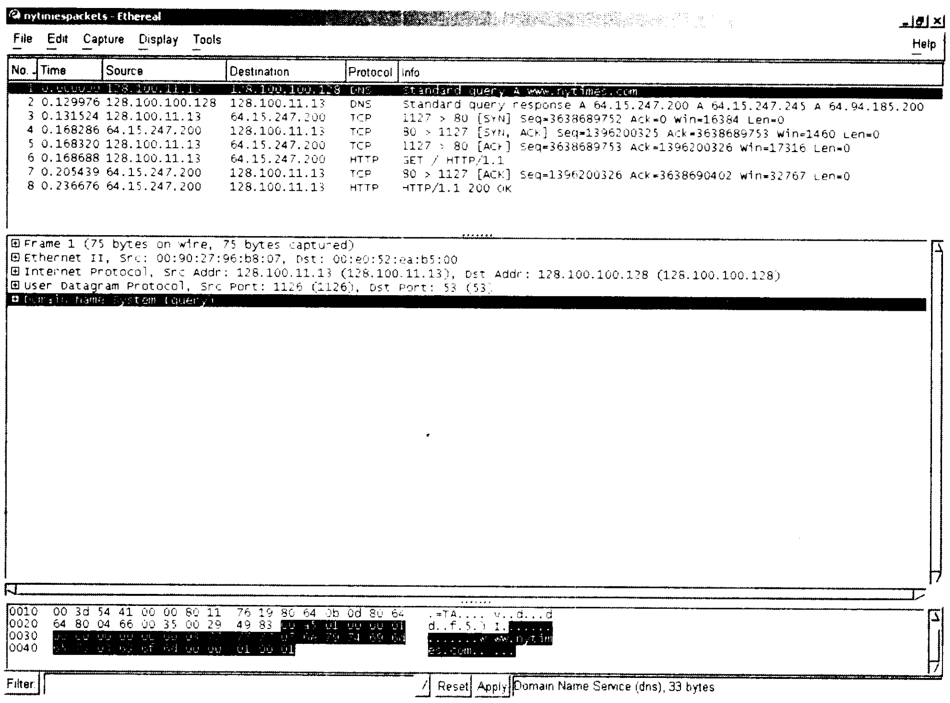
VIEWING THE LAYERS USING A NETWORK PROTOCOL ANALYZER

A network protocol analyzer is a tool that can capture, display, and analyze the PDUs that are exchanged between peer processes. Protocol analyzers are extremely useful in troubleshooting network problems and also as an educational tool. Network protocol analyzers are discussed in the last section of this chapter. In our examples we will use the *Ethereal* open source package. In this section we use a sequence of captured packets to show how the layers work together in a simple web interaction.

Figure 2.18 shows the Ethereal display after capturing packets that are transmitted after clicking on the URL of the *New York Times*. The top pane in the display shows the first eight packets that are transmitted during the interaction:

1. The first packet carries a DNS query from the machine with IP address 128.100.100.13 for the IP address of www.nytimes.com. It can be seen from the first packet that the IP address of the local DNS server is 128.100.100.128. The second packet carries the DNS response that provides three IP addresses, 64.15.347.200, 64.15.347.245, and 64.94.185.200 for the URL.
2. The next three packets correspond to the *three-way handshake* that is used to establish a TCP connection between the client and the server.[9] In the first packet the client (with IP address 128.100.100.128 and port address 1127) makes a TCP connection setup request by sending an IP packet to 64.15.347.200 and well-known port number 80. In the first packet, the client also includes an initial sequence number to keep

[9]TCP is discussed in detail in Chapter 8.



```
No. Time     Source          Destination     Protocol Info
  1 0.000000 128.100.11.13   128.100.100.128 DNS      Standard query A www.nytimes.com
  2 0.129976 128.100.100.128 128.100.11.13   DNS      Standard query response A 64.15.247.200 A 64.15.247.245 A 64.94.185.200
  3 0.131524 128.100.11.13   64.15.247.200   TCP      1127 > 80 [SYN] Seq=3638689752 Ack=0 Win=16384 Len=0
  4 0.168286 64.15.247.200   128.100.11.13   TCP      80 > 1127 [SYN, ACK] Seq=1396200325 Ack=3638689753 Win=1460 Len=0
  5 0.168320 128.100.11.13   64.15.247.200   TCP      1127 > 80 [ACK] Seq=3638689753 Ack=1396200326 Win=17316 Len=0
  6 0.168688 128.100.11.13   64.15.247.200   HTTP     GET / HTTP/1.1
  7 0.205439 64.15.247.200   128.100.11.13   TCP      80 > 1127 [ACK] Seq=1396200326 Ack=3638690402 Win=32767 Len=0
  8 0.236676 64.15.247.200   128.100.11.13   HTTP     HTTP/1.1 200 OK

⊞ Frame 1 (75 bytes on wire, 75 bytes captured)
⊞ Ethernet II, Src: 00:90:27:96:b8:07, Dst: 00:e0:52:ea:b5:00
⊞ Internet Protocol, Src Addr: 128.100.11.13 (128.100.11.13), Dst Addr: 128.100.100.128 (128.100.100.128)
⊞ User Datagram Protocol, Src Port: 1126 (1126), Dst Port: 53 (53)
⊞ Domain Name System (query)

0010  00 3d 54 41 00 00 80 11  76 19 80 64 0b 0d 80 64   .=TA.... v..d...d
0020  64 80 04 66 00 35 00 29  49 83 00 a5 01 00 00 01   d..f.5.) I.......
0030  00 00 00 00 00 00 03 77  77 77 07 6e 79 74 69 6d   .......w ww.nytim
0040  65 73 03 63 6f 6d 00 00  01 00 01                  es.com.. ...

Domain Name Service (dns), 33 bytes
```

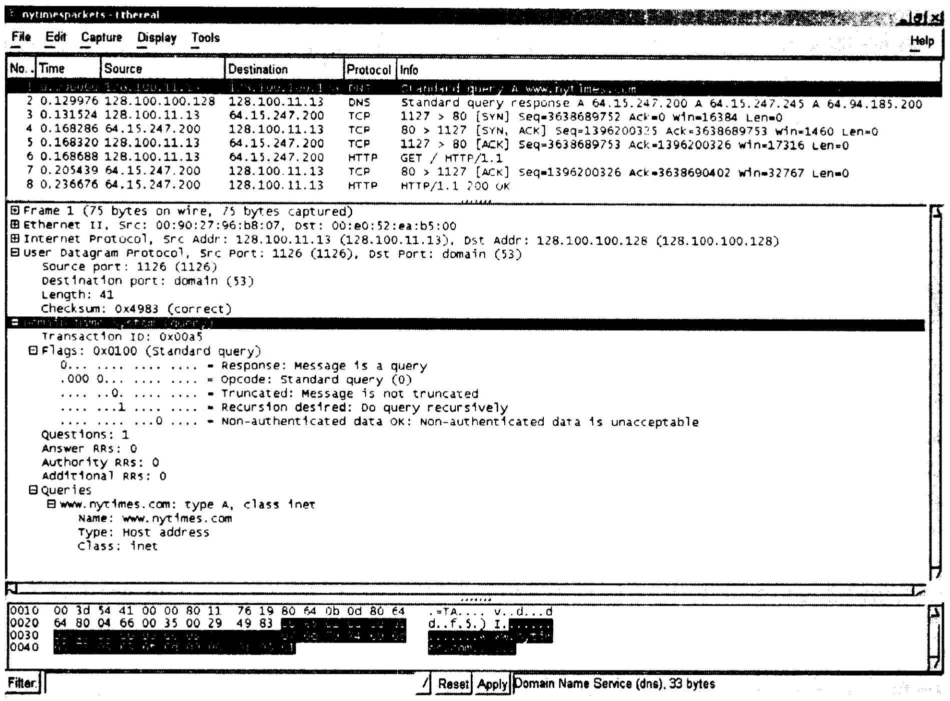
FIGURE 2.18 Viewing packet exchanges and protocol layers using Ethereal: the display has three panes (from top to bottom): packet capture list, details of selected packet, and data from the selected packet.

count of the bytes it transmits. In the second packet the server acknowledges the connection request and proposes its own initial sequence number. With the third packet, the client confirms the TCP connection setup and the initial sequence numbers. The TCP connection is now ready.

3. The sixth packet carries the HTTP "GET" request from the client to the server.
4. The seventh packet carries an acknowledgment message that is part of the TCP protocol.
5. The final packet carries the HTTP status response from the server. The response code 200 confirms that the request was successful and that the document will follow.

The process of encapsulation (see Figure 2.17) means that a given captured frame carries information from multiple layers. Figure 2.18 illustrates this point quite clearly. The middle pane displays information about the highlighted packet (a DNS query) of the top pane. By looking down the list in the middle pane, one can see the protocol stack that the DNS query traversed, UDP over IP over Ethernet.

Figure 2.19 provides more information about the same DNS packet (obtained by clicking on the "+" to the left of the desired entry in the middle pane). In the figure the UDP and DNS entries have been expanded. In the UDP entry, it can be seen that the first packet carries source port number 1126 and well-known destination port number 53. The DNS entry shows the contents of the DNS query. Finally the third pane in Figure 2.19


nytimespackets - Ethereal
File Edit Capture Display Tools Help
Filter: Reset Apply

```
No. Time     Source          Destination     Protocol Info
  1 [illegible] 128.100.11.13 128.100.100.128 DNS  Standard query A www.nytimes.com
  2 0.129976 128.100.100.128 128.100.11.13   DNS      Standard query response A 64.15.247.200 A 64.15.247.245 A 64.94.185.200
  3 0.131524 128.100.11.13   64.15.247.200   TCP      1127 > 80 [SYN] Seq=3638689752 Ack=0 Win=16384 Len=0
  4 0.168286 64.15.247.200   128.100.11.13   TCP      80 > 1127 [SYN, ACK] Seq=1396200325 Ack=3638689753 Win=1460 Len=0
  5 0.168320 128.100.11.13   64.15.247.200   TCP      1127 > 80 [ACK] Seq=3638689753 Ack=1396200326 Win=17316 Len=0
  6 0.168688 128.100.11.13   64.15.247.200   HTTP     GET / HTTP/1.1
  7 0.205439 64.15.247.200   128.100.11.13   TCP      80 > 1127 [ACK] Seq=1396200326 Ack=3638690402 Win=32767 Len=0
  8 0.236676 64.15.247.200   128.100.11.13   HTTP     HTTP/1.1 200 OK

⊞ Frame 1 (75 bytes on wire, 75 bytes captured)
⊞ Ethernet II, Src: 00:90:27:96:b8:07, Dst: 00:e0:52:ea:b5:00
⊞ Internet Protocol, Src Addr: 128.100.11.13 (128.100.11.13), Dst Addr: 128.100.100.128 (128.100.100.128)
⊟ User Datagram Protocol, Src Port: 1126 (1126), Dst Port: domain (53)
    Source port: 1126 (1126)
    Destination port: domain (53)
    Length: 41
    Checksum: 0x4983 (correct)
⊟ Domain Name System (query)
    Transaction ID: 0x00a5
  ⊟ Flags: 0x0100 (Standard query)
      0... .... .... .... = Response: Message is a query
      .000 0... .... .... = Opcode: Standard query (0)
      .... ..0. .... .... = Truncated: Message is not truncated
      .... ...1 .... .... = Recursion desired: Do query recursively
      .... .... ...0 .... = Non-authenticated data OK: Non-authenticated data is unacceptable
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  ⊟ Queries
    ⊟ www.nytimes.com: type A, class inet
        Name: www.nytimes.com
        Type: Host address
        Class: inet

0010  00 3d 54 41 00 00 80 11  76 19 80 64 0b 0d 80 64   .=TA.... v..d...d
0020  64 80 04 66 00 35 00 29  49 83 [illegible]          d..f.5.) I.[illegible]
0030  [illegible]                                         [illegible]
0040  [illegible]                                         [illegible]

Domain Name Service (dns), 33 bytes
```

FIGURE 2.19 More detailed protocol layer information for selected captured packet.

shows the actual raw data of the captured packet. The highlighted data in the third pane corresponds to the fields that are highlighted in the middle pane. Thus the highlighted area in the figure contains the data relating to the DNS PDU. From the third line in the middle pane we can see the source and destination IP addresses of the IP datagram that carries the given UDP packet. The second line shows the Ethernet physical addresses of this frame that carries the given UDP packet. By expanding the IP and Ethernet entries we can obtain the details of the IP datagram and the Ethernet frame. We will explore the details of these and other protocols in the remainder of the book.

2.3.3 Protocol Overview

This completes the discussion on how the different layers work together in a TCP/IP Internet. In the remaining chapters we examine the details of the operation of the various layers. In Chapters 3 and 4 we consider various aspects of physical layers. In Chapter 5 we discuss peer-to-peer protocols that allow protocols such as TCP to provide reliable service. We also discuss data link control protocols. In Chapter 6 we discuss LANs and their medium access controls. In Chapter 7 we return to the network layer and examine the operation of routers and packet switches as well as issues relating to addressing, routing, and congestion control. Chapter 8 presents a detailed discussion of the TCP and IP protocols. In Chapter 9 we introduce ATM, a connection-oriented packet network architecture. In Chapter 10 we discuss advanced topics, such as connection-oriented IP networks realized through MPLS, new developments in TCP/IP architecture and the support of real-time multimedia services over IP. In Chapter 11 we introduce enhancements to IP that provide security. From time to time it may be worthwhile to return to this example to place the discussion of details in the subsequent chapters into the big picture presented here.

◆ 2.4 THE BERKELEY API[10]

An Application Programming Interface (API) allows application programs (such as Telnet, web browsers, etc.) to access certain resources through a predefined and preferably consistent interface. One of the most popular of the APIs that provide access to network resources is the Berkeley socket interface, which was developed by a group at the University of California at Berkeley in the early 1980s. The socket interface is now widely available on many UNIX machines. Another popular socket interface, which was derived from the Berkeley socket interface, is called the Windows sockets or Winsock and was designed to operate in a Microsoft® Windows environment.

By hiding the details of the underlying communication technologies as much as possible, the socket mechanism allows programmers to write application programs easily without worrying about the underlying networking details. Figure 2.20 shows how two applications talk to each other across a communication network through the socket interface. In a typical communication session, one application operates as a server and the other as a client. The server is the provider of a particular service while the client is the consumer. A server waits passively most of the time until a client requires a service.

This section explains how the socket mechanism can provide services to the applications. Two modes of services are available through the socket interface: *connection-oriented* and *connectionless*. With the connection-oriented mode, an application must first establish a connection to the other end before the actual communication (i.e., data transfer) can take place. The connection is established if the other end agrees to accept the connection. Once the connection is established, data will be delivered through the connection to the destination in sequence. The connection-oriented mode provides a reliable delivery service. With the connectionless mode an application sends its data immediately without waiting for the connection to get established at all. This mode avoids the setup overhead found in the connection-oriented mode. However, the price to pay is that an application may waste its time sending data when the other end is not ready to accept it. Moreover, data may not arrive at the other end if the network decides to discard it. Worse yet, even if data arrives at the destination, it may not arrive in the same order as it was transmitted. The connectionless mode is said to provide *best-effort service,* since the network would try its best to deliver the information but cannot guarantee delivery.

Figure 2.21 shows a typical diagram of the sequence of socket calls for the connection-oriented mode. The server begins by carrying out a *passive open* as

[10]This section is optional and is not required for later sections. A knowledge of C programming is assumed.

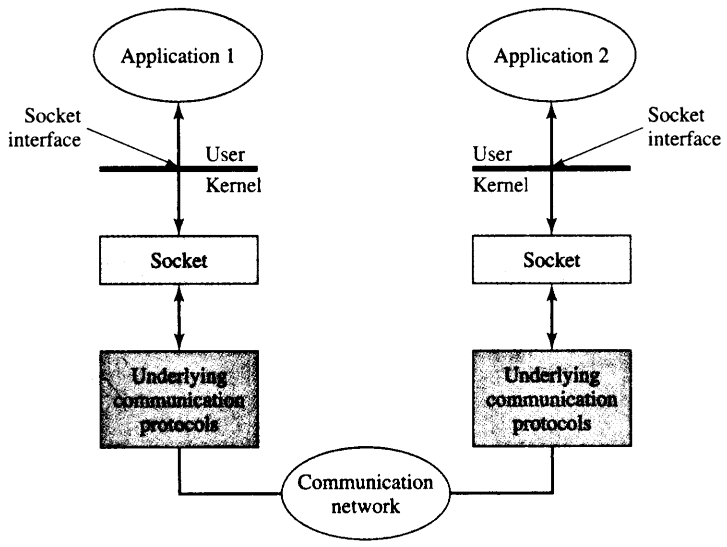


FIGURE 2.20 Communications through the socket interface.

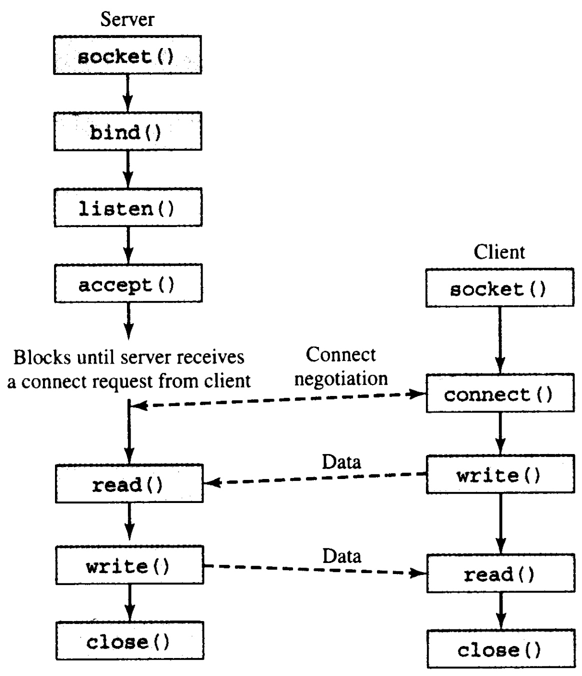


FIGURE 2.21 Socket calls for connection-oriented mode.

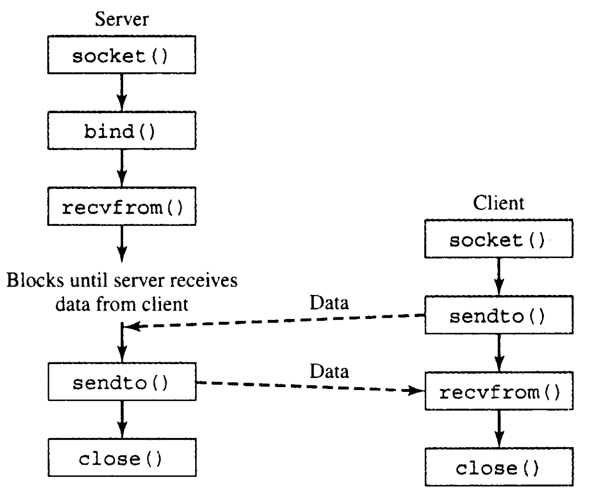


FIGURE 2.22 Socket calls for connectionless mode.

follows. The `socket` call creates a TCP socket. The `bind` call then binds the well-known port number of the server to the socket. The `listen` call turns the socket into a listening socket that can accept incoming connections from clients. Finally, the `accept` call puts the server process to sleep until the arrival of a client connection request. The client does an *active open*. The `socket` call creates a socket on the client side, and the `connect` call attempts to establish the TCP connection to the server with the specified destination socket address. When the TCP connection is established, the `accept` function at the server wakes up and returns the descriptor for the given connection, namely, the source IP address, source port number, destination IP address, and destination port number. The client and server are now ready to exchange information.

Figure 2.22 shows the sequence of socket calls for the connectionless mode. Note that no connection is established prior to data transfer. The `recvfrom` call returns when a complete UDP datagram has been received. For both types of communication, the data transfer phase may occur in an arbitrary number of exchanges.

2.4.1 Socket System Calls

Socket facilities are provided to programmers through C system calls that are similar to function calls except that control is transferred to the operating system kernel once a call is entered. To use these facilities, the header files `<sys/types.h>` and `<sys/socket.h>` must be included in the program.

CREATING A SOCKET

Before an application program (client or server) can transfer any data, it must first create an endpoint for communication by calling `socket`. Its prototype is

```
int socket(int family, int type, int protocol);
```

where family identifies the family by address or protocol. The address family identifies a collection of protocols with the same address format, while the protocol family identifies a collection of protocols having the same architecture. Although it may be possible to classify the family based on addresses or protocols, these two families are currently equivalent. Some examples of the address family that are defined in <sys/socket.h> include AF_UNIX, which is used for communication on the local UNIX machine, and AF_INET, which is used for Internet communication using TCP/IP protocols. The protocol family is identified by the prefix PF_. The value of PF_XXX is equal to that of AF_XXX, indicating that the two families are equivalent. We are concerned only with AF_INET in this book.

The type identifies the semantics of communication. Some of the types include SOCK_STREAM, SOCK_DGRAM, and SOCK_RAW. A SOCK_STREAM type provides data delivery service as a sequence of bytes and does not preserve message boundaries. A SOCK_DGRAM type provides data delivery service in blocks of bytes called datagrams. A SOCK_RAW type provides access to internal network interfaces and is available only to superuser.

The protocol identifies the specific protocol to be used. Normally, only one protocol is available for each family and type, so the value for the protocol argument is usually set to 0 to indicate the default protocol. The default protocol of SOCK_STREAM type with AF_INET family is TCP, which is a connection-oriented protocol providing a reliable service with in-sequence data delivery. The default protocol of SOCK_DGRAM type with AF_INET family is UDP, which is a connectionless protocol with unreliable service.

The socket call returns a nonnegative integer value called the socket descriptor or handle (just like a file descriptor) on success. On failure, socket returns −1.

ASSIGNING AN ADDRESS TO THE SOCKET

After a socket is created, the bind system call can be used to assign an address to the socket. Its prototype is

```
int bind(int sd, struct sockaddr *name, int namelen);
```

where sd is the socket descriptor returned by the socket call, name is a pointer to an address structure that contains the local IP address and port number, and namelen is the size of the address structure in bytes. The bind system call returns 0 on success and −1 on failure. The sockaddr structure is a generic address structure and has the following definition:

```
struct sockaddr {
        u_short  sa_family;          /* address family  */
        char     sa_data[14];        /* address   */
};
```

where sa_family holds the address family and sa_data holds up to 14 bytes of address information that varies from one family to another. For the Internet family the address information consists of the port number that is two bytes long and an IP address that is four bytes long. The appropriate structures to use for the Internet family are defined in <netinet/in.h>:

```
struct in addr {
        u_long    s_addr;              /* 32-bit IP address */
};
struct sockaddr_in {
        u_short    sin_family;          /* AF_INET */
        u_short    sin_port;            /* TCP or UDP port */
        struct     in_addr sin_addr;    /* 32-bit IP address */
        char       sin_zero[8];         /* unused */
};
```

An application program using the Internet family should use the sockaddr_in structure to assign member values and should use the sockaddr structure only for casting purposes in function arguments. For this family sin_family holds the value of the identifier AF_INET. The structure member sin_port holds the local port number. Port numbers 1 to 1023 are normally reserved for system use. For a server, sin_port contains a well-known port number that clients must know in advance to establish a connection. Specifying a port number 0 to bind asks the system to assign an available port number. The structure member sin_addr holds the local IP address. For a host with multiple IP addresses, sin_addr is typically set to INADDR_ANY to indicate that the server is willing to accept communication through any of its IP addresses. This setting is useful for a host with multiple IP addresses. The structure member sin_zero is used to fill out struct sockaddr_in to 16 bytes.

Different computers may store a multibyte word in different orders. If the least significant byte is stored first (has lower address), it is known as *little endian.* If the most significant byte is stored first, it is known as *big endian.* For any two computers to be able to communicate, they must agree on a common data format while transferring multibyte words. The Internet adopts the big-endian format. This representation is known as *network byte order* in contrast to the representation adopted by the host, which is called *host byte order.* It is important to remember that the values of sin_port and sin_addr must be in the network byte order, since these values are communicated across the network. Four functions are available to convert between the host and network byte order conveniently. Functions htons and htonl convert an unsigned short and an unsigned long, respectively, from the host to network byte order. Functions ntohs and ntohl convert an unsigned short and an unsigned long, respectively, from the network to host byte order. We need to use these functions so that programs will be portable to any machine. To use these functions, we should include the header files <sys/types.h> and <netinet/in.h>. The appropriate prototypes are

```
u_long htonl(u_long hostlong);
u_short htons(u_short hostshort);
u_long ntohl(u_long netlong);
u_short ntohs(u_short netshort);
```

ESTABLISHING AND ACCEPTING CONNECTIONS

A client establishes a connection on a socket by calling connect. The prototype is

```
int connect(int sd, struct sockaddr *name, int namelen);
```

where `sd` is the socket descriptor returned by the `socket` call, `name` points to the server address structure, and `namelen` specifies the amount of space in bytes pointed to by `name`. For the connection-oriented mode, `connect` attempts to establish a connection between a client and a server. For the connectionless mode, `connect` stores the server's address so that the client can use a mode socket descriptor when sending datagrams, instead of specifying the server's address each time a datagram is sent. The `connect` system call returns 0 on success and −1 on failure.

A connection-oriented server indicates its willingness to receive connection requests by calling `listen`. The prototype is

```
int listen(int sd, int backlog);
```

where `sd` is the socket descriptor returned by the `socket` call and `backlog` specifies the maximum number of connection requests that the system should queue while it waits for the server to accept them (the maximum value is usually 5). This mechanism allows pending connection requests to be saved while the server is busy processing other tasks. The `listen` system call returns 0 on success and −1 on failure.

After a server calls `listen`, it can accept the connection request by calling `accept` with the prototype

```
int accept(int sd, struct sockaddr *addr, int *addrlen);
```

where `sd` is the socket descriptor returned by the `socket` call, `addr` is a pointer to an address structure that `accept` fills in with the client's IP address and port number, and `addrlen` is a pointer to an integer specifying the amount of space pointed to by `addr` before the call. On return, the value pointed to by `addrlen` specifies the number of bytes of the client address information.

If no connection requests are pending, `accept` will block the caller until a connection request arrives. The `accept` system call returns a new socket descriptor having nonnegative value on success and −1 on failure. The new socket descriptor inherits the properties of `sd`. The server uses the new socket descriptor to perform data transfer for the new connection. While data transfer occurs on an existing connection, a *concurrent* server can accept further connection requests using the original socket descriptor `sd`, allowing multiple clients to be served simultaneously.

TRANSMITTING AND RECEIVING DATA

Clients and servers may transmit data using `write` or `sendto`. The `write` call is usually used for the connection-oriented mode. However, a connectionless client may also call `write` if it has a connected socket (that is, the client has executed `connect`). On the other hand, the `sendto` call is usually used for the connectionless mode. Their prototypes are

```
int write(int sd, char *buf, int buflen);
int sendto(int sd, char *buf, int buflen, int flags,
   struct sockaddr *addrp, int addrlen);
```

where `sd` is the socket descriptor, `buf` is a pointer to a buffer containing the data to transmitted, `buflen` is the length of the data in bytes, `flags` can be used to control transmission behavior such as handling out-of-band (high priority) data but is usually set to 0 for normal operation, addrp is a pointer to the sockaddr structure containing the address information of the remote hosts, and addrlen is the length of the address information. Both write and sendto return the number of bytes transmitted on success or −1 on failure.

The corresponding system calls to receive data read and recvfrom. Their prototypes are

```
int read(int sd, char *buf, int buflen);
int recvfrom(int sd, char * buf, int buflen, int flags,
   struct sockaddr *addrp, int *addrlen);
```

The parameters are similar to the ones discussed above except buf is now a pointer to a buffer that is used to store the received data and buflen is the length of the buffer in bytes. Both read and recvfrom return the number of bytes received on success or −1 on failure. Both calls will block if no data arrives at the local host.

CLOSING A CONNECTION

If a socket is no longer in use, the application can call close to terminate a connection and return system resources to the operating system. The prototype is

```
int close(int sd);
```

where sd is the socket descriptor to be closed. The close call returns 0 on success and −1 on failure.

2.4.2 Network Utility Functions

Library routines are available to convert a human-friendly domain name such as tesla.comm.utoronto.ca into a 32-bit machine-friendly IP as 10000000 01100100 00001011 00000001 and vice versa. To perform the conversion we should include the header files <sys/socket.h>, <sys/types.h>, and <netdb.h>. The appropriate structure that stores the host information defined in the <netdb.h> file is

```
struct hostent {
    char *h_name;               /* official name of host */
    char **h_aliases;           /* alias name this host uses */
    int h_addrtype;             /* address type */
    int h_length;               /* length of address */
    char **h_addr_list;         /* list of addresses from name
                                   server */
};
```

The h_name element points to the official name of the host. If the host has name aliases, these aliases are pointed to by h_aliases, which is terminated by a NULL. Thus h_aliases[0] points to the first alias, h_aliases[1] points to the second alias, and so on. Currently, the `h_addrtype` element always takes on the value of `AF_INET`, and the `h_length` element always contains a value of 4. The `h_addr_list` points to the list of network addresses in network byte order and is terminated by a `NULL`.

NAME-TO-ADDRESS CONVERSION FUNCTIONS

Two functions are used for routines performing a name-to-address-conversion: `gethostbyname` and `gethostbyaddr`.

```
struct hostent *gethostbyname (char *name);
```

The function `gethostbyname` takes a domain name at the input and returns the host information as a pointer to `struct hostent`. The function returns a `NULL` on error. The parameter `name` is a pointer to a domain name of a host whose information we would like to obtain. The function `gethostbyname` obtains the host information either from the file `/etc/hosts` or from a name server. Recall that the host information includes the desired address.

```
struct hostent *gethostbyaddr (char *addr, int len, int type);
```

The function `gethostbyaddr` takes a host address at the input in network byte order, its length in bytes, and type, which should be `AF_INET`. The function returns the same information as `gethostbyname`. This information includes the desired host name.

The IP address is usually communicated by people using a notation called the *dotted-decimal notation.* As an example, the dotted-decimal notation of the IP address 10000000 01100100 00001011 00000001 is 128.100.11.1. To convert between these two formats, we could use the functions `inet_addr` and `inet_ntoa`. The header files that must be included are `<sys/types.h>`, `<sys/socket.h>`, `<netinet/in.h>`, and `<arpa/inet.h>`.

IP ADDRESS MANIPULATION FUNCTIONS

Two functions are used for routines converting addresses between a 32-bit format and the dotted-decimal notation: `inet_nota and inet_addr`.

```
char *inet_ntoa(struct in_addr in);
```

The function `inet_ntoa` takes a 32-bit IP address in network byte order and returns the corresponding address in dotted-decimal notation.

```
unsigned long inet_addr(char *cp);
```

The function `inet_addr` takes a host address in dotted-decimal notation and returns the corresponding 32-bit IP address in network byte order.

EXAMPLE Communicating with TCP

As an illustration of the use of the system calls and functions described previously, let us show two application programs that communicate via TCP. The client prompts a user to type a line of text, sends it to the server, reads the data back from the server, and prints it out. The server acts as a simple echo server. After responding to a client, the server closes the connection and then waits for the next new connection. In this example each application (client and server) expects a fixed number of bytes from the other end, specified by `BUFLEN`. Because TCP is stream oriented, the received data may come in multiple pieces of byte streams independent of how the data was sent at the other end. For example, when a transmitter sends 100 bytes of data in a single `write` call, the receiver may receive the data in two pieces—80 bytes and 20 bytes—or in three pieces—10 bytes, 50 bytes, and 40 bytes—or in any other combination. Thus the program has to make repeated calls to `read` until all the data has been received. The following program is the server.

```
/* A simple echo server using TCP */
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define SERVER_TCP_PORT   3000        /* well-known port */
#define BUFLEN            256         /* buffer length */

int main(int argc, char **argv)
{
    int      n, bytes_to_read;
    int      sd, new_sd, client_len, port;
    struct   sockaddr_in server, client;
    char     *bp, buf[BUFLEN];

    switch(argc) {
    case 1:
        port = SERVER_TCP_PORT;
        break;
    case 2:
        port = atoi(argv[1]);
        break;
    default:
        fprintf(stderr, "Usage: %s [port]\n", argv[0]);
        exit(1);
    }

    /* Create a stream socket */
    if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        fprintf(stderr, "Can't create a socket\n");
        exit(1);
    }
    /* Bind an address to the socket */
    bzero((char *)&server, sizeof(struct sockaddr_in));
    server.sin_family = AF_INET;
    server.sin_port = htons(port);
    server.sin_addr.s_addr = htonl(INADDR_ANY);
    if (bind(sd, (struct sockaddr *)&server,
    sizeof(server)) == -1) {
      fprintf(stderr, "Can't bind name to socket\n");
      exit(1);
    }

    /* queue up to 5 connect requests */
    listen(sd, 5);

     while (1) {
      client_len = sizeof(client);
      if ((new_sd = accept(sd, (struct sockaddr *)
      &client, &client_len)) == -1) {
        fprintf(stderr, "Can't accept client\n");
        exit(1);
      }

      bp = buf;
        bytes_to_read = BUFLEN;
        while ((n = read(new_sd, bp, bytes_to_read)) > 0) {
          bp += n;
          bytes_to_read -= n;
        }

        write(new_sd, buf, BUFLEN);
        close(new_sd);
      }
      close(sd);
      return(0);
}
```

The client program allows the user to identify the server by its domain name. Conversion to the IP address is done by the `gethostbyname` function. Again, the client makes repeated calls to `read` until no more data is expected to arrive. The following program is the client.

```
/* A simple TCP client */
#include <stdio.h>
#include <netdb.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define SERVER_TCP_PORT   3000
#define BUFLEN         256      /* buffer length */
```

```
int main(int argc, char **argv)
{
    int      n, bytes_to_read;
    int      sd, port;
    struct   hostent      *hp;
    struct   sockaddr_in  server;
    char     *host, *bp, rbuf[BUFLEN], sbuf[BUFLEN];

    switch(argc) {
    case 2:
        host = argv[1];
        port = SERVER_TCP_PORT;
        break;
    case 3:
        host = argv[1];
        port = atoi(argv[2]);
        break;
    default:
        fprintf(stderr, "Usage: %s host[port]\n", argv[0]);
        exit(1);
    }

    /* Create a stream socket */
    if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
         fprintf(stderr, "Can't create a socket\n");
         exit(1);
    }

    bzero((char *)&server, sizeof(struct sockaddr_in));
    server.sin_family = AF_INET;
    server.sin_port = htons(port);
    if ((hp = gethostbyname(host)) == NULL) {
         fprintf(stderr, "Can't get server's address\n");
         exit(1);
     }
     bcopy(hp->h_addr, (char *)&server.sin_addr,
         hp->h_length);

     /* Connecting to the server */
     if (connect(sd, (struct sockaddr *)&server,
     sizeof(server)) == -1) {
         fprintf(stderr, "Can't connect\n");
         exit(1);
     }
     printf("Connected: server's address is %s\n",
         hp->h_name);
    printf("Transmit:\n");
    gets(sbuf);          /* get user's text */
    write(sd, sbuf, BUFLEN); /* send it out */

    printf("Receive:\n");
    bp = rbuf;
    bytes_to_read = BUFLEN;
    while ((n = read (sd, bp, bytes_to_read)) > 0) {
        bp += n;
        bytes_to_read -= n;
    }
    printf("%s\n", rbuf);

    close(sd);
    return(0);
}
```

The student is encouraged to verify the sequence of socket calls in the above client and server programs with those shown in Figure 2.21. Further, the student may trace the sequence of calls by inserting a print statement after each call and verify that the `accept` call in the TCP server blocks until the `connect` call in the TCP client returns.

EXAMPLE Using the UDP Protocol

Let us now take a look at client/server programs using the UDP protocol. The following source code is a program that uses the UDP server as an echo server as before. Note that data receipt can be done in a single call with `recvfrom`, since UDP is blocked oriented.

```
/* Echo server using UDP */
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define SERVER_UDP_PORT  5000     /* well-known port */
#define MAXLEN           4096     /* maximum data length */

int main(int argc, char **argv)
{
   int      sd, client_len, port, n;
   char     buf[MAXLEN];
   struct   sockaddr_in server, client;
```

```
    switch(argc) {
    case 1:
         port = SERVER_UDP_PORT;
         break;
    case 2:
         port = atoi(argv[1]);
         break;
    default:
         fprintf(stderr, "Usage: %s [port]\n", argv[0]);
         exit(1);
    }

    /* Create a datagram socket */
    if ((sd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
         fprintf(stderr, "Can't create a socket\n");
         exit(1);
    }

    /* Bind an address to the socket */
    bzero((char *)&server, sizeof(server));
    server.sin_family = AF_INET;
    server.sin_port = htons(port);
    server.sin_addr.s_addr = htonl(INADDR_ANY);
    if (bind(sd, (struct sockaddr *)&server,
    sizeof(server)) == -1) {
         fprintf(stderr, "Can't bind name to socket\n");
         exit(1);
    }

    while (1) {
         client_len = sizeof(client);
         if ((n = recvfrom(sd, buf, MAXLEN, 0,
         (struct sockaddr *)&client, &client_len)) < 0) {
             fprintf(stderr, "Can't receive datagram\n");
             exit(1);
         }

         if (sendto(sd, buf, n, 0,
         (struct sockaddr *)&client, client_len) != n) {
              fprintf(stderr, "Can't send datagram\n");
              exit(1);
         }
    }
    close(sd);
    return(0);
}
```

The following client program first constructs a simple message of a predetermined length containing a string of characters a, b, c, . . . , z, a, b, c, . . . , z, . . . The client then gets the `start` time from the system using `gettimeofday` and sends the message to the echo server. After the message travels back, the client records the end time and measures the difference that represents the round-trip latency between the client and the server. The unit of time is recorded in milliseconds. This simple example shows how we can use sockets to gather important network statistics such as latencies and jitter.

```
/* A simple UDP client which measures round trip delay */
#include <stdio.h>
#include <string.h>
#include <sys/time.h>
#include <netdb.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define SERVER_UDP_PORT  5000
#define MAXLEN       4096     /* maximum data length */
#define DEFLEN       64       /* default length */

long delay(struct timeval t1, struct timeval t2);

int main(int argc, char **argv)
{
      int       data_size = DEFLEN, port = SERVER_UDP_PORT;
      int       i, j, sd, server_len;
      char      *pname, *host, rbuf[MAXLEN], sbuf[MAXLEN];
      struct    hostent    *hp;
      struct    sockaddr_in     server;
      struct    timeval      start, end;

      pname = argv[0];
      argc--;
      argv++;
      if (argc > 0 && (strcmp(*argv, "-s") == 0)) {
          if (--argc > 0 && (data_size = atoi(*++argv))) {
               argc--;
               argv++;
          }
          else {
               fprintf (stderr,
               "Usage: %s [-s data_size] host [port]\n",
               pname);
               exit(1);
          }
      }
      if (argc > 0) {
          host = *argv;
          if (--argc > 0)
              port = atoi(*++argv);
      }
```

```
else {
    fprintf(stderr,
    "Usage: %s [-s data_size] host [port]\n", pname);
    exit(1);
}

/* Create a datagram socket */
if ((sd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
    fprintf(stderr, "Can't create a socket\n");
    exit(1);
}

/* Store server's information */
bzero((char *)&server, sizeof(server));
server.sin_family = AF_INET;
server.sin_port = htons(port);
if ((hp = gethostbyname(host)) == NULL) {
    fprintf(stderr, "Can't get server's IP address\n");
    exit(1);
}
bcopy(hp->h_addr, (char *)&server.sin_addr,
     hp->h_length);
if (data_size > MAXLEN) {
     fprintf(stderr, "Data is too big\n");
     exit(1);
}
/* data is a, b, c,..., z, a, b,... */
for (i = 0; i < data_size; i++) {
     j = (i < 26) ? i : i % 26;
     sbuf[i] = 'a' + j;
}

gettimeofday(&start, NULL); /* start delay measure */

/* transmit data */
server_len = sizeof(server);
if (sendto(sd, sbuf, data_size, 0, (struct sockaddr *)
     &server, server_len) == -1) {
     fprintf(stderr, "sendto error\n");
     exit(1);
}

/* receive data */
if (recvfrom(sd, rbuf, MAXLEN, 0, (struct sockaddr *)
     &server, &server_len) < 0) {
     fprintf(stderr, "recvfrom error\n");
     exit(1);
}
```

```
    gettimeofday(&end, NULL); /* end delay measure */

    printf ("Round-trip delay = %ld ms.\n",
         delay(start, end));

    if (strncmp(sbuf, rbuf, data_size) != 0)
        printf("Data is corrupted\n");

    close(sd);
    return(0);
}

/*
 * Compute the delay between t1 and t2 in milliseconds
 */
long delay (struct timeval t1, struct timeval t2)
{
     long d;

     d = (t2.tv_sec - t1.tv_sec) * 1000;
     d += ((t2.tv_usec - t1.tv_usec + 500) / 1000);
     return(d);
}
```

It is important to remember that datagram communication using UDP is unreliable. If the communication is restricted to a local area network environment, say within a building, then datagram losses are extremely rare in practice, and the above client program should work well. However, in a wide area network environment, datagrams may be frequently discarded by the network. If the reply from the server does not reach the client, the client will wait forever! In this situation, the client must provide a timeout mechanism and retransmit the message. Also, further reliability may be provided to reorder the datagram at the receiver and to ensure that duplicated datagrams are discarded.

◆ 2.5 APPLICATION LAYER PROTOCOLS AND TCP/IP UTILITIES

Application layer protocols are high-level protocols that provide services to user applications. These protocols tend to be more visible to the user than other types of protocols. Furthermore, application protocols may be user written, or they may be standardized applications. Several standard application protocols form part of the TCP/IP protocol suite, the more common ones being Telnet, File Transfer Protocol (FTP), HTTP, and SMTP. Coverage of the various TCP/IP application layer protocols is beyond the scope of this textbook. The student is referred to "Internet Official Protocol Standards," which provides a list of Internet protocols and standards [RFC 3000]. In this section the focus is on applications and utilities that can be used as tools to study the operation of the Internet. We also introduce network protocol analyzers and explain the basics of packet capture.

2.5.1 Telnet

Telnet is a TCP/IP protocol that provides a standardized means of accessing resources on a remote machine where the initiating machine is treated as local to the remote host. In many implementations Telnet can be used to connect to the port number of other servers and to interact with them using a command line. For example, the HTTP and SMTP examples in Section 2.1 were generated this way.

The Telnet protocol is based on the concept of a *network virtual terminal (NVT),* which is an imaginary device that represents a lowest common denominator terminal. By basing the protocol on this interface, the client and server machines do not have to obtain information about each other's terminal characteristics. Instead, each machine initially maps its characteristics to that of an NVT and *negotiates options* for changes to the NVT or other enhancements, such as changing the character set.

The NVT acts as a character-based terminal with a keyboard and printer. Data input by the client through the keyboard is sent to the server through the Telnet connection. This data is echoed back by the server to the client's printer. Other incoming data from the server is also printed.

Telnet commands use the seven-bit U.S. variant of the ASCII character set. A command consists minimally of a two-byte sequence: the Interpret as Command (IAC) escape character followed by the command code. If the command pertains to option negotiation, that is, one of WILL, WONT, DO, or DONT, then a third byte contains the option code. Table 2.4 lists the Telnet command names, their corresponding ASCII code, and their meaning.

A substantial number of Telnet options can be negotiated. Option negotiations begin once the connection is established and may occur at any time while connected. Negotiation is symmetric in the sense that either side can initiate a negotiation. A negotiation syntax is defined in RFC 854 to prevent acknowledgment loops from occurring.

Telnet uses one TCP connection. Because a TCP connection is identified by a pair of port numbers, a server is capable of supporting more than one Telnet connection at a time. Once the connection is established, the default is for the user, that is, the initiator of the connection, to enter a login name and password. By default the password is sent as clear text, although more recent versions of Telnet offer an authentication option.

2.5.2 File Transfer Protocol

File Transfer Protocol (FTP) is another commonly used application protocol. FTP provides for the transfer of a file from one machine to another. Like Telnet, FTP is intended to operate across different hosts, even when they are running different operating systems or have different file structures.