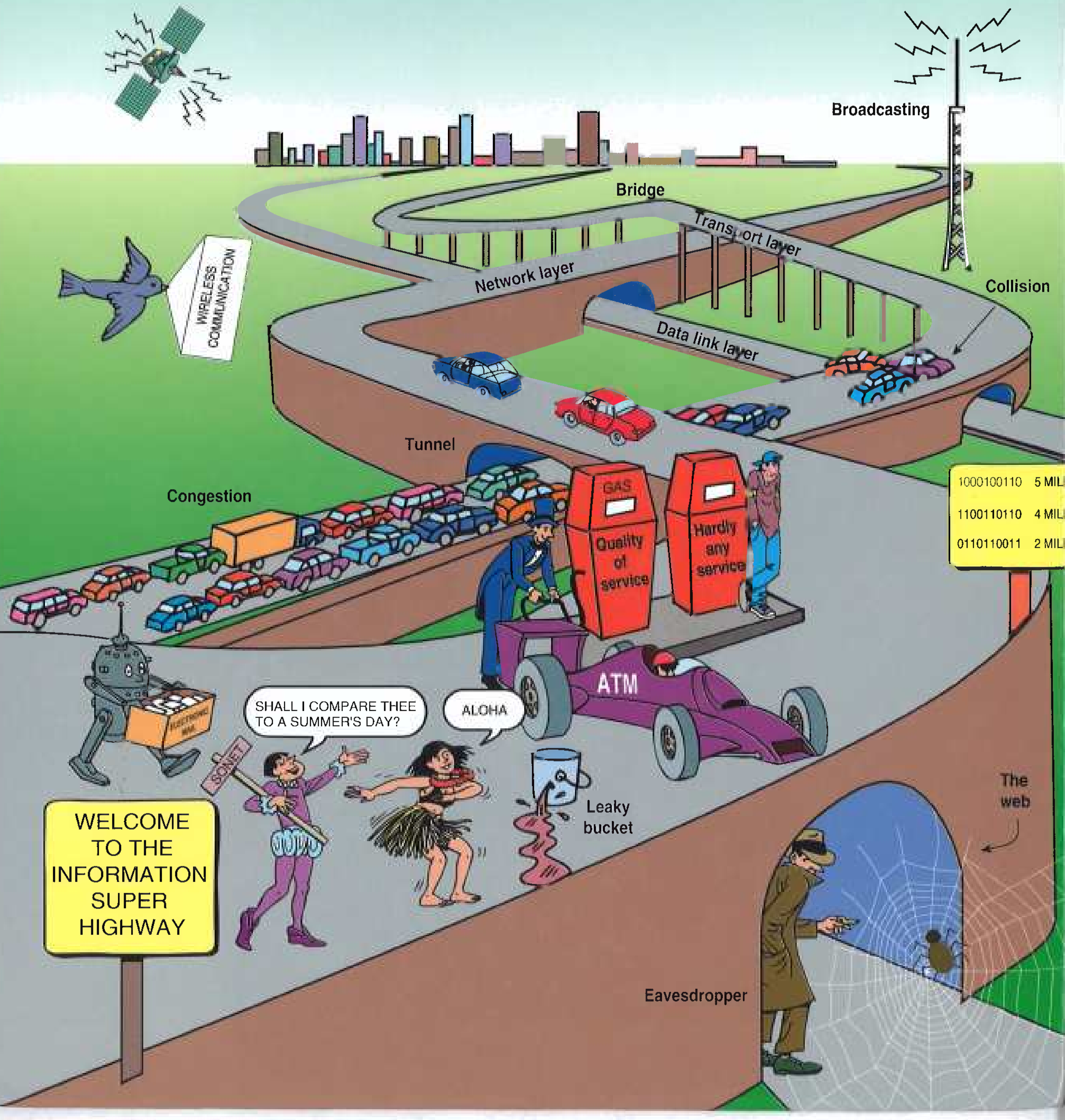
THIRD EDITION

COMPUTER NETWORKS

ANDREW S. TANENBAUM



Library of Congress Cataloging in Publication Data

Tanenbaum, Andrew S. 1944-.
Computer networks / Andrew S. Tanenbaum. -- 3rd ed.
p. cm.
Includes bibliographical references and index.
ISBN 0-13-349945-6
1.Computer networks. I. Title.
TK5105.5.T36 1996 96-4121
004.6--dc20 CIP

Editorial/production manager: *Camille Trentacoste*
Interior design and composition: *Andrew S. Tanenbaum*
Cover design director: *Jerry Votta*
Cover designer: *Don Martinetti, DM Graphics, Inc.*
Cover concept: *Andrew S. Tanenbaum, from an idea by Marilyn Tremaine*
Interior graphics: *Hadel Studio*
Manufacturing manager: *Alexis R. Heydt*
Acquisitions editor: *Mary Franz*
Editorial Assistant: *Noreen Regina*

Prentice-Hall, Inc.
A Simon & Schuster Company
Upper Saddle River, New Jersey 07458

The publisher offers discounts on this book when ordered in bulk quantities. For more information, contact:
Corporate Sales Department, Prentice Hall PTR, One Lake Street, Upper Saddle River, NJ 07458.
Phone: (800) 382-3419; Fax: (201) 236-7141. E-mail: corpsales@prenhall.com

Printed in the United States of America
10 9 8 7 6 5 4

ISBN 0-13-349945-6

Prentice-Hall International (UK) Limited, *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Prentice-Hall Canada Inc., *Toronto*
Prentice-Hall Hispanoamericana, S.A., *Mexico*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Simon & Schuster Asia Pte. Ltd., *Singapore*
Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*

CONTENTS

2 THE PHYSICAL LAYER 77

4 THE MEDIUM ACCESS SUBLAYER 243

5 THE NETWORK LAYER 339

6 THE TRANSPORT LAYER 479

7 THE APPLICATION LAYER 577

computers, while large institutions had at most a few dozen. The idea that within 20 years equally powerful computers smaller than postage stamps would be mass produced by the millions was pure science fiction.

The merging of computers and communications has had a profound influence on the way computer systems are organized. The concept of the "computer center" as a room with a large computer to which users bring their work for processing is now totally obsolete. The old model of a single computer serving all of the organization's computational needs has been replaced by one in which a large number of separate but interconnected computers do the job. These systems are called **computer networks**. The design and organization of these networks are the subjects of this book.

Throughout the book we will use the term "computer network" to mean an *interconnected* collection of *autonomous* computers. Two computers are said to be interconnected if they are able to exchange information. The connection need not be via a copper wire; fiber optics, microwaves, and communication satellites can also be used. By requiring the computers to be autonomous, we wish to exclude from our definition systems in which there is a clear master/slave relation. If one computer can forcibly start, stop, or control another one, the computers are not autonomous. A system with one control unit and many slaves is not a network; nor is a large computer with remote printers and terminals.

There is considerable confusion in the literature between a computer network and a **distributed system**. The key distinction is that in a distributed system, the existence of multiple autonomous computers is transparent (i.e., not visible) to the user. He[†] can type a command to run a program, and it runs. It is up to the operating system to select the best processor, find and transport all the input files to that processor, and put the results in the appropriate place.

In other words, the user of a distributed system is not aware that there are multiple processors; it looks like a virtual uniprocessor. Allocation of jobs to processors and files to disks, movement of files between where they are stored and where they are needed, and all other system functions must be automatic.

With a network, users must *explicitly* log onto one machine, *explicitly* submit jobs remotely, *explicitly* move files around and generally handle all the network management personally. With a distributed system, nothing has to be done explicitly; it is all automatically done by the system without the users' knowledge.

In effect, a distributed system is a software system built on top of a network. The software gives it a high degree of cohesiveness and transparency. Thus the distinction between a network and a distributed system lies with the software (especially the operating system), rather than with the hardware.

Nevertheless, there is considerable overlap between the two subjects. For example, both distributed systems and computer networks need to move files around. The difference lies in who invokes the movement, the system or the user.

[†] "He" should be read as "he or she" throughout this book.

Although this book primarily focuses on networks, many of the topics are also important in distributed systems. For more information about distributed systems, see (Coulouris et al., 1994; Mullender, 1993; and Tanenbaum, 1995).

1.1. USES OF COMPUTER NETWORKS

Before we start to examine the technical issues in detail, it is worth devoting some time to pointing out why people are interested in computer networks and what they can be used for.

1.1.1. Networks for Companies

Many organizations have a substantial number of computers in operation, often located far apart. For example, a company with many factories may have a computer at each location to keep track of inventories, monitor productivity, and do the local payroll. Initially, each of these computers may have worked in isolation from the others, but at some point, management may have decided to connect them to be able to extract and correlate information about the entire company.

Put in slightly more general form, the issue here is **resource sharing**, and the goal is to make all programs, equipment, and especially data available to anyone on the network without regard to the physical location of the resource and the user. In other words, the mere fact that a user happens to be 1000 km away from his data should not prevent him from using the data as though they were local. This goal may be summarized by saying that it is an attempt to end the "tyranny of geography."

A second goal is to provide **high reliability** by having alternative sources of supply. For example, all files could be replicated on two or three machines, so if one of them is unavailable (due to a hardware failure), the other copies could be used. In addition, the presence of multiple CPUs means that if one goes down, the others may be able to take over its work, although at reduced performance. For military, banking, air traffic control, nuclear reactor safety, and many other applications, the ability to continue operating in the face of hardware problems is of utmost importance.

Another goal is **saving money**. Small computers have a much better price/performance ratio than large ones. Mainframes (room-size computers) are roughly a factor of ten faster than personal computers, but they cost a thousand times more. This imbalance has caused many systems designers to build systems consisting of personal computers, one per user, with data kept on one or more shared **file server** machines. In this model, the users are called **clients**, and the whole arrangement is called the **client-server model**. It is illustrated in Fig. 1-1.

In the client-server model, communication generally takes the form of a request message from the client to the server asking for some work to be done.

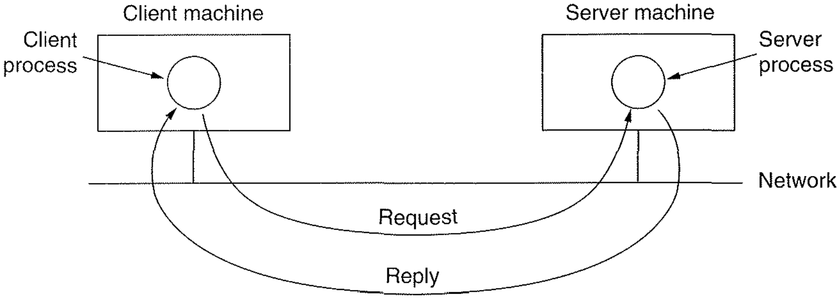


Fig. 1-1. The client-server model.

The server then does the work and sends back the reply. Usually, there are many clients using a small number of servers.

Another networking goal is scalability, the ability to increase system performance gradually as the workload grows just by adding more processors. With centralized mainframes, when the system is full, it must be replaced by a larger one, usually at great expense and even greater disruption to the users. With the client-server model, new clients and new servers can be added as needed.

Yet another goal of setting up a computer network has little to do with technology at all. A computer network can provide a powerful **communication medium** among widely separated employees. Using a network, it is easy for two or more people who live far apart to write a report together. When one worker makes a change to an on-line document, the others can see the change immediately, instead of waiting several days for a letter. Such a speedup makes cooperation among far-flung groups of people easy where it previously had been impossible. In the long run, the use of networks to enhance human-to-human communication will probably prove more important than technical goals such as improved reliability.

1.1.2. Networks for People

The motivations given above for building computer networks are all essentially economic and technological in nature. If sufficiently large and powerful mainframes were available at acceptable prices, most companies would simply choose to keep all their data on them and give employees terminals connected to them. In the 1970s and early 1980s, most companies operated this way. Computer networks only became popular when networks of personal computers offered a huge price/performance advantage over mainframes.

Starting in the 1990s, computer networks began to start delivering services to private individuals at home. These services and the motivations for using them are quite different than the "corporate efficiency" model described in the previous section. Below we will sketch three of the more exciting ones that are starting to happen:

1. Access to remote information.
2. Person-to-person communication.
3. Interactive entertainment.

Access to remote information will come in many forms. One area in which it is already happening is access to financial institutions. Many people pay their bills, manage their bank accounts, and handle their investments electronically. Home shopping is also becoming popular, with the ability to inspect the on-line catalogs of thousands of companies. Some of these catalogs will soon provide the ability to get an instant video on any product by just clicking on the product's name.

Newspapers will go on-line and be personalized. It will be possible to tell the newspaper that you want everything about corrupt politicians, big fires, scandals involving celebrities, and epidemics, but no football, thank you. At night while you sleep, the newspaper will be downloaded to your computer's disk or printed on your laser printer. On a small scale, this service already exists. The next step beyond newspapers (plus magazines and scientific journals) is the on-line digital library. Depending on the cost, size, and weight of book-sized notebook computers, printed books may become obsolete. Skeptics should take note of the effect the printing press had on the medieval illuminated manuscript.

Another application that falls in this category is access to information systems like the current World Wide Web, which contains information about the arts, business, cooking, government, health, history, hobbies, recreation, science, sports, travel, and too many other topics to even mention.

All of the above applications involve interactions between a person and a remote database. The second broad category of network use will be person-to-person interactions, basically the 21st Century's answer to the 19th Century's telephone. Electronic mail or **email** is already widely used by millions of people and will soon routinely contain audio and video as well as text. Smell in messages will take a bit longer to perfect.

Real-time email will allow remote users to communicate with no delay, possibly seeing and hearing each other as well. This technology makes it possible to have virtual meetings, called **videoconference**, among far-flung people. It is sometimes said that transportation and communication are having a race, and whichever wins will make the other obsolete. Virtual meetings could be used for remote school, getting medical opinions from distant specialists, and numerous other applications.

Worldwide newsgroups, with discussions on every conceivable topic are already commonplace among a select group of people, and this will grow to include the population at large. These discussions, in which one person posts a message and all the other subscribers to the newsgroup can read it, run the gamut from humorous to impassioned.

Our third category is entertainment, which is a huge and growing industry. The killer application here (the one that may drive all the rest) is video on demand. A decade or so hence, it may be possible to select any movie or television program ever made, in any country, and have it displayed on your screen instantly. New films may become interactive, where the user is occasionally prompted for the story direction (should Macbeth murder Duncan or just bide his time?) with alternative scenarios provided for all cases. Live television may also become interactive, with the audience participating in quiz shows, choosing among contestants, and so on.

On the other hand, maybe the killer application will not be video on demand. Maybe it will be game playing. Already we have multiperson real-time simulation games, like hide-and-seek in a virtual dungeon, and flight simulators with the players on one team trying to shoot down the players on the opposing team. If done with goggles and 3-dimensional real-time, photographic-quality moving images, we have a kind of worldwide shared virtual reality.

In short, the ability to merge information, communication, and entertainment will surely give rise to a massive new industry based on computer networking.

1.1.3. Social Issues

The widespread introduction of networking will introduce new social, ethical, political problems (Laudon, 1995). Let us just briefly mention a few of them; a thorough study would require a full book, at least. A popular feature of many networks are newsgroups or bulletin boards where people can exchange messages with like-minded individuals. As long as the subjects are restricted to technical topics or hobbies like gardening, not too many problems will arise.

The trouble comes when newsgroups are set up on topics that people actually care about, like politics, religion, or sex. Views posted to such groups may be deeply offensive to some people. Furthermore, messages need not be limited to text. High-resolution color photographs and even short video clips can now easily be transmitted over computer networks. Some people take a live-and-let-live view, but others feel that posting certain material (e.g., child pornography) is simply unacceptable. Thus the debate rages.

People have sued network operators, claiming that they are responsible for the contents of what they carry, just as newspapers and magazines are. The inevitable response is that a network is like a telephone company or the post office and cannot be expected to police what its users say. Stronger yet, having network operators censor messages would probably cause them to delete everything with even the slightest possibility of their being sued, and thus violate their users' rights to free speech. It is probably safe to say that this debate will go on for a while.

Another fun area is employee rights versus employer rights. Many people read and write email at work. Some employers have claimed the right to read and possibly censor employee messages, including messages sent from a home terminal after work. Not all employees agree with this (Sipior and Ward, 1995).

Even if employers have power over employees, does this relationship also govern universities and students? How about high schools and students? In 1994, Carnegie-Mellon University decided to turn off the incoming message stream for several newsgroups dealing with sex because the university felt the material was inappropriate for minors (i.e., those few students under 18). The fallout from this event will take years to settle.

Computer networks offer the potential for sending anonymous messages. In some situations, this capability may be desirable. For example, it provides a way for students, soldiers, employees, and citizens to blow the whistle on illegal behavior on the part of professors, officers, superiors, and politicians without fear of reprisals. On the other hand, in the United States and most other democracies, the law specifically permits an accused person the right to confront and challenge his accuser in court. Anonymous accusations cannot be used as evidence.

In short, computer networks, like the printing press 500 years ago, allow ordinary citizens to distribute their views in different ways and to different audiences than were previously possible. This new-found freedom brings with it many unsolved social, political, and moral issues. The solution to these problems is left as an exercise for the reader.

1.2. NETWORK HARDWARE

It is now time to turn our attention from the applications and social aspects of networking to the technical issues involved in network design. There is no generally accepted taxonomy into which all computer networks fit, but two dimensions stand out as important: transmission technology and scale. We will now examine each of these in turn.

Broadly speaking, there are two types of transmission technology:

1. Broadcast networks.
2. Point-to-point networks.

Broadcast networks have a single communication channel that is shared by all the machines on the network. Short messages, called **packets** in certain contexts, sent by any machine are received by all the others. An address field within the packet specifies for whom it is intended. Upon receiving a packet, a machine checks the address field. If the packet is intended for itself, it processes the packet; if the packet is intended for some other machine, it is just ignored.

As an analogy, consider someone standing at the end of a corridor with many rooms off it and shouting "Watson, come here. I want you." Although the packet are free to change their protocols at will, provided they do not change the service visible to their users. In this way, the service and the protocol are completely decoupled.

An analogy with programming languages is worth making. A service is like an abstract data type or an object in an object-oriented language. It defines operations that can be performed on an object but does not specify how these operations are implemented. A protocol relates to the *implementation* of the service and as such is not visible to the user of the service.

Many older protocols did not distinguish the service from the protocol. In effect, a typical layer might have had a service primitive SEND PACKET with the user providing a pointer to a fully assembled packet. This arrangement meant that all changes to the protocol were immediately visible to the users. Most network designers now regard such a design as a serious blunder.

1.4. REFERENCE MODELS

Now that we have discussed layered networks in the abstract, it is time to look at some examples. In the next two sections we will discuss two important network architectures, the OSI reference model and the TCP/IP reference model.

1.4.1. The OSI Reference Model

The OSI model is shown in Fig. 1-16 (minus the physical medium). This model is based on a proposal developed by the International Standards Organization (ISO) as a first step toward international standardization of the protocols used in the various layers (Day and Zimmermann, 1983). The model is called the **ISO OSI (Open Systems Interconnection) Reference Model** because it deals with connecting open systems—that is, systems that are open for communication with other systems. We will usually just call it the OSI model for short.

The OSI model has seven layers. The principles that were applied to arrive at the seven layers are as follows:

1. A layer should be created where a different level of abstraction is needed.
2. Each layer should perform a well defined function.
3. The function of each layer should be chosen with an eye toward defining internationally standardized protocols.
4. The layer boundaries should be chosen to minimize the information flow across the interfaces.
5. The number of layers should be large enough that distinct functions need not be thrown together in the same layer out of necessity, and small enough that the architecture does not become unwieldy.

Below we will discuss each layer of the model in turn, starting at the bottom layer. Note that the OSI model itself is not a network architecture because it does not specify the exact services and protocols to be used in each layer. It just tells what each layer should do. However, ISO has also produced standards for all the layers, although these are not part of the reference model itself. Each one has been published as a separate international standard.

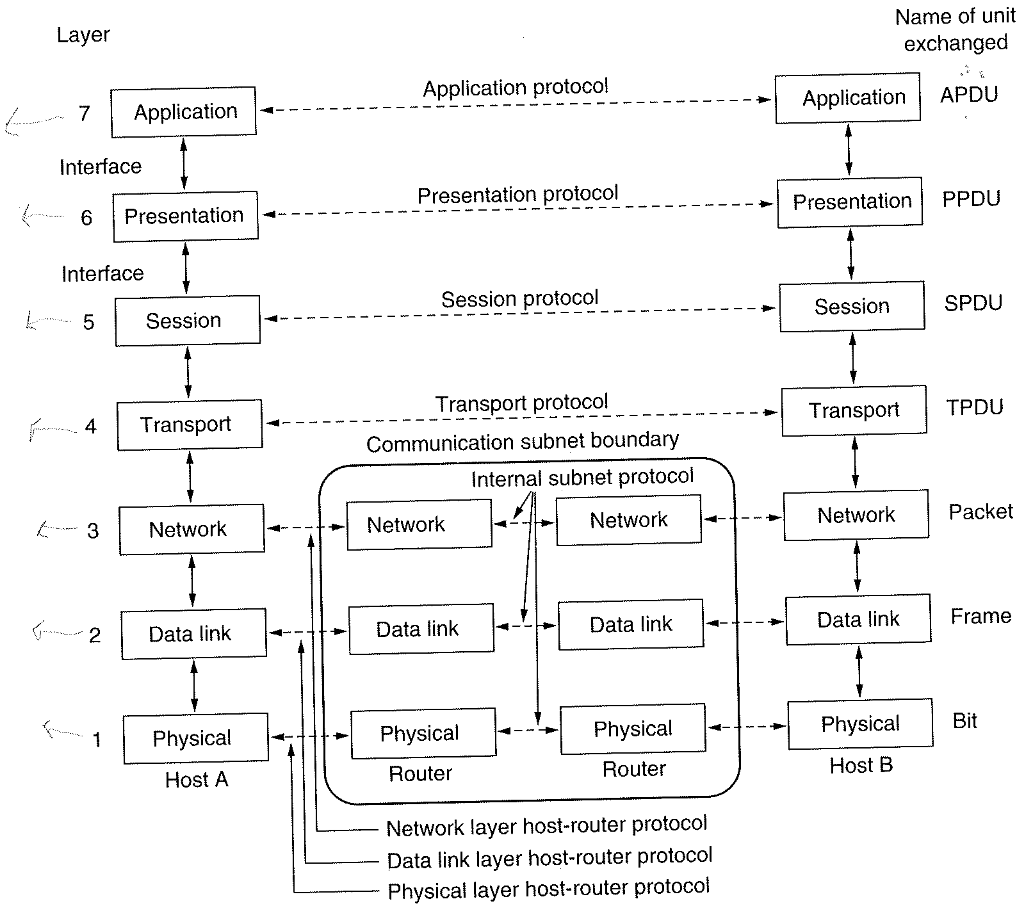


Fig. 1-16. The OSI reference model.

The Physical Layer

The **physical layer** is concerned with transmitting raw bits over a communication channel. The design issues have to do with making sure that when one side sends a 1 bit, it is received by the other side as a 1 bit, not as a 0 bit. Typical questions here are how many volts should be used to represent a 1 and how many for a 0, how many microseconds a bit lasts, whether transmission may proceed simultaneously in both directions, how the initial connection is established and how it is torn down when both sides are finished, and how many pins the network connector has and what each pin is used for. The design issues here largely deal with mechanical, electrical, and procedural interfaces, and the physical transmission medium, which lies below the physical layer.

The Data Link Layer

The main task of the **data link layer** is to take a raw transmission facility and transform it into a line that appears free of undetected transmission errors to the network layer. It accomplishes this task by having the sender break the input data up into **data frames** (typically a few hundred or a few thousand bytes), transmit the frames sequentially, and process the **acknowledgement frames** sent back by the receiver. Since the physical layer merely accepts and transmits a stream of bits without any regard to meaning or structure, it is up to the data link layer to create and recognize frame boundaries. This can be accomplished by attaching special bit patterns to the beginning and end of the frame. If these bit patterns can accidentally occur in the data, special care must be taken to make sure these patterns are not incorrectly interpreted as frame delimiters.

A noise burst on the line can destroy a frame completely. In this case, the data link layer software on the source machine can retransmit the frame. However, multiple transmissions of the same frame introduce the possibility of duplicate frames. A duplicate frame could be sent if the acknowledgement frame from the receiver back to the sender were lost. It is up to this layer to solve the problems caused by damaged, lost, and duplicate frames. The data link layer may offer several different service classes to the network layer, each of a different quality and with a different price.

Another issue that arises in the data link layer (and most of the higher layers as well) is how to keep a fast transmitter from drowning a slow receiver in data. Some traffic regulation mechanism must be employed to let the transmitter know how much buffer space the receiver has at the moment. Frequently, this flow regulation and the error handling are integrated.

If the line can be used to transmit data in both directions, this introduces a new complication that the data link layer software must deal with. The problem is that the acknowledgement frames for *A* to *B* traffic compete for the use of the line with data frames for the *B* to *A* traffic. A clever solution (piggybacking) has been devised; we will discuss it in detail later.

Broadcast networks have an additional issue in the data link layer: how to control access to the shared channel. A special sublayer of the data link layer, the medium access sublayer, deals with this problem.

The Network Layer

The **network layer** is concerned with controlling the operation of the subnet. A key design issue is determining how packets are routed from source to destination. Routes can be based on static tables that are "wired into" the network and rarely changed. They can also be determined at the start of each conversation, for example a terminal session. Finally, they can be highly dynamic, being determined anew for each packet, to reflect the current network load.

If too many packets are present in the subnet at the same time, they will get in each other's way, forming bottlenecks. The control of such congestion also belongs to the network layer.

Since the operators of the subnet may well expect remuneration for their efforts, there is often some accounting function built into the network layer. At the very least, the software must count how many packets or characters or bits are sent by each customer, to produce billing information. When a packet crosses a national border, with different rates on each side, the accounting can become complicated.

When a packet has to travel from one network to another to get to its destination, many problems can arise. The addressing used by the second network may be different from the first one. The second one may not accept the packet at all because it is too large. The protocols may differ, and so on. It is up to the network layer to overcome all these problems to allow heterogeneous networks to be interconnected.

In broadcast networks, the routing problem is simple, so the network layer is often thin or even nonexistent.

The Transport Layer

The basic function of the **transport layer** is to accept data from the session layer, split it up into smaller units if need be, pass these to the network layer, and ensure that the pieces all arrive correctly at the other end. Furthermore, all this must be done efficiently, and in a way that isolates the upper layers from the inevitable changes in the hardware technology.

Under normal conditions, the transport layer creates a distinct network connection for each transport connection required by the session layer. If the transport connection requires a high throughput, however, the transport layer might create multiple network connections, dividing the data among the network connections to improve throughput. On the other hand, if creating or maintaining a network connection is expensive, the transport layer might multiplex several transport connections onto the same network connection to reduce the cost. In all cases, the transport layer is required to make the multiplexing transparent to the session layer.

The transport layer also determines what type of service to provide the session layer, and ultimately, the users of the network. The most popular type of transport connection is an error-free point-to-point channel that delivers messages or bytes in the order in which they were sent. However, other possible kinds of transport service are transport of isolated messages with no guarantee about the order of delivery, and broadcasting of messages to multiple destinations. The type of service is determined when the connection is established.

The transport layer is a true end-to-end layer, from source to destination. In other words, a program on the source machine carries on a conversation with a similar program on the destination machine, using the message headers and control messages. In the lower layers, the protocols are between each machine and its immediate neighbors, and not by the ultimate source and destination machines, which may be separated by many routers. The difference between layers 1 through 3, which are chained, and layers 4 through 7, which are end-to-end, is illustrated in Fig. 1-16.

Many hosts are multiprogrammed, which implies that multiple connections will be entering and leaving each host. There needs to be some way to tell which message belongs to which connection. The transport header (H_4 in Fig. 1-11) is one place this information can be put.

In addition to multiplexing several message streams onto one channel, the transport layer must take care of establishing and deleting connections across the network. This requires some kind of naming mechanism, so that a process on one machine has a way of describing with whom it wishes to converse. There must also be a mechanism to regulate the flow of information, so that a fast host cannot overrun a slow one. Such a mechanism is called **flow control** and plays a key role in the transport layer (also in other layers). Flow control between hosts is distinct from flow control between routers, although we will later see that similar principles apply to both.

The Session Layer

The session layer allows users on different machines to establish **sessions** between them. A session allows ordinary data transport, as does the transport layer, but it also provides enhanced services useful in some applications. A session might be used to allow a user to log into a remote timesharing system or to transfer a file between two machines.

One of the services of the session layer is to manage dialogue control. Sessions can allow traffic to go in both directions at the same time, or in only one direction at a time. If traffic can only go one way at a time (analogous to a single railroad track), the session layer can help keep track of whose turn it is.

A related session service is **token management**. For some protocols, it is essential that both sides do not attempt the same operation at the same time. To manage these activities, the session layer provides tokens that can be exchanged. Only the side holding the token may perform the critical operation.

Another session service is **synchronization**. Consider the problems that might occur when trying to do a 2-hour file transfer between two machines with a 1-hour mean time between crashes. After each transfer was aborted, the whole transfer would have to start over again and would probably fail again the next time as well. To eliminate this problem, the session layer provides a way to insert checkpoints into the data stream, so that after a crash, only the data transferred after the last checkpoint have to be repeated.

The Presentation Layer

The **presentation layer** performs certain functions that are requested sufficiently often to warrant finding a general solution for them, rather than letting each user solve the problems. In particular, unlike all the lower layers, which are just interested in moving bits reliably from here to there, the presentation layer is concerned with the syntax and semantics of the information transmitted.

A typical example of a presentation service is encoding data in a standard agreed upon way. Most user programs do not exchange random binary bit strings. They exchange things such as people's names, dates, amounts of money, and invoices. These items are represented as character strings, integers, floating-point numbers, and data structures composed of several simpler items. Different computers have different codes for representing character strings (e.g., ASCII and Unicode), integers (e.g., one's complement and two's complement), and so on. In order to make it possible for computers with different representations to communicate, the data structures to be exchanged can be defined in an abstract way, along with a standard encoding to be used "on the wire." The presentation layer manages these abstract data structures and converts from the representation used inside the computer to the network standard representation and back.

The Application Layer

The **application layer** contains a variety of protocols that are commonly needed. For example, there are hundreds of incompatible terminal types in the world. Consider the plight of a full screen editor that is supposed to work over a network with many different terminal types, each with different screen layouts, escape sequences for inserting and deleting text, moving the cursor, etc.

One way to solve this problem is to define an abstract **network virtual terminal** that editors and other programs can be written to deal with. To handle each terminal type, a piece of software must be written to map the functions of the network virtual terminal onto the real terminal. For example, when the editor moves the virtual terminal's cursor to the upper left-hand corner of the screen, this software must issue the proper command sequence to the real terminal to get its cursor there too. All the virtual terminal software is in the application layer.

Another application layer function is file transfer. Different file systems have different file naming conventions, different ways of representing text lines, and so on. Transferring a file between two different systems requires handling these and other incompatibilities. This work, too, belongs to the application layer, as do electronic mail, remote job entry, directory lookup, and various other general-purpose and special-purpose facilities.

Data Transmission in the OSI Model

Figure 1-17 shows an example of how data can be transmitted using the OSI model. The sending process has some data it wants to send to the receiving process. It gives the data to the application layer, which then attaches the application header, *AH* (which may be null), to the front of it and gives the resulting item to the presentation layer.

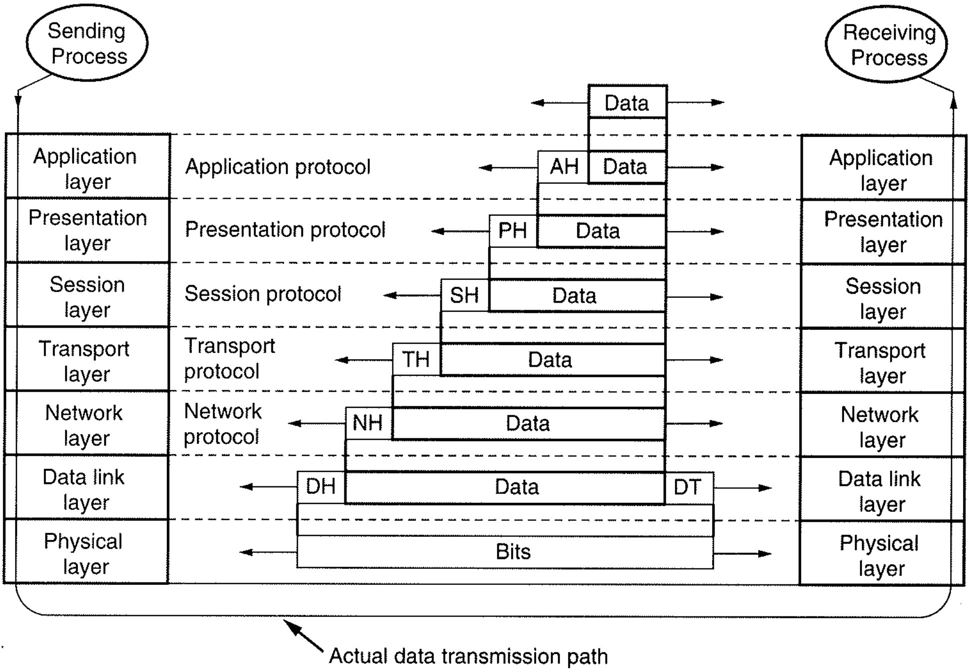


Fig. 1-17. An example of how the OSI model is used. Some of the headers may be null. (Source: H.C. Folts. Used with permission.)

The presentation layer may transform this item in various ways and possibly add a header to the front, giving the result to the session layer. It is important to realize that the presentation layer is not aware of which portion of the data given to it by the application layer is *AH*, if any, and which is true user data.

This process is repeated until the data reach the physical layer, where they are actually transmitted to the receiving machine. On that machine the various headers are stripped off one by one as the message propagates up the layers until it finally arrives at the receiving process.

The key idea throughout is that although actual data transmission is vertical in Fig. 1-17, each layer is programmed as though it were horizontal. When the sending transport layer, for example, gets a message from the session layer, it attaches a transport header and sends it to the receiving transport layer. From its point of view, the fact that it must actually hand the message to the network layer on its own machine is an unimportant technicality. As an analogy, when a Tagalog-speaking diplomat is addressing the United Nations, he thinks of himself as addressing the other assembled diplomats. That, in fact, he is really only speaking to his translator is seen as a technical detail.

1.4.2. The TCP/IP Reference Model

Let us now turn from the OSI reference model to the reference model used in the grandparent of all computer networks, the ARPANET, and its successor, the worldwide Internet. Although we will give a brief history of the ARPANET later, it is useful to mention a few key aspects of it now. The ARPANET was a research network sponsored by the DoD (U.S. Department of Defense). It eventually connected hundreds of universities and government installations using leased telephone lines. When satellite and radio networks were added later, the existing protocols had trouble interworking with them, so a new reference architecture was needed. Thus the ability to connect multiple networks together in a seamless way was one of the major design goals from the very beginning. This architecture later became known as the **TCP/IP Reference Model**, after its two primary protocols. It was first defined in (Cerf and Kahn, 1974). A later perspective is given in (Leiner et al., 1985). The design philosophy behind the model is discussed in (Clark, 1988).

Given the DoD's worry that some of its precious hosts, routers, and internetwork gateways might get blown to pieces at a moment's notice, another major goal was that the network be able to survive loss of subnet hardware, with existing conversations not being broken off. In other words, DoD wanted connections to remain intact as long as the source and destination machines were functioning, even if some of the machines or transmission lines in between were suddenly put out of operation. Furthermore, a flexible architecture was needed, since applications with divergent requirements were envisioned, ranging from transferring files to real-time speech transmission.

The Internet Layer

All these requirements led to the choice of a packet-switching network based on a connectionless internetwork layer. This layer, called the **internet layer**, is the linchpin that holds the whole architecture together. Its job is to permit hosts to inject packets into any network and have them travel independently to the destination (potentially on a different network). They may even arrive in a different order than they were sent, in which case it is the job of higher layers to rearrange them, if in-order delivery is desired. Note that "internet" is used here in a generic sense, even though this layer is present in the Internet.

The analogy here is with the (snail) mail system. A person can drop a sequence of international letters into a mail box in one country, and with a little luck, most of them will be delivered to the correct address in the destination country. Probably the letters will travel through one or more international mail gateways along the way, but this is transparent to the users. Furthermore, that each country (i.e., each network) has its own stamps, preferred envelope sizes, and delivery rules is hidden from the users.

The internet layer defines an official packet format and protocol called **IP (Internet Protocol)**. The job of the internet layer is to deliver IP packets where they are supposed to go. Packet routing is clearly the major issue here, as is avoiding congestion. For these reasons, it is reasonable to say that the TCP/IP internet layer is very similar in functionality to the OSI network layer. Figure 1-18 shows this correspondence.

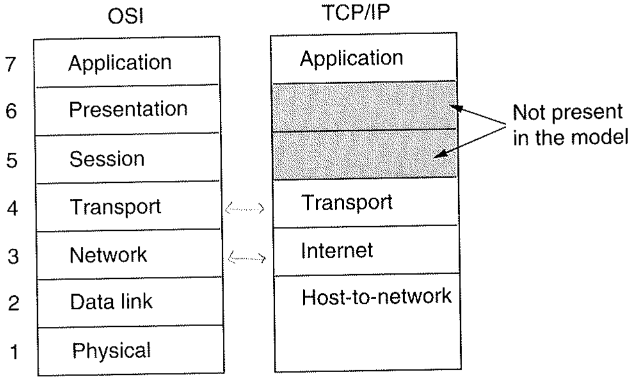


Fig. 1-18. The TCP/IP reference model.

The Transport Layer

The layer above the internet layer in the TCP/IP model is now usually called the **transport layer**. It is designed to allow peer entities on the source and destination hosts to carry on a conversation, the same as in the OSI transport layer. Two end-to-end protocols have been defined here. The first one, **TCP (Transmission Control Protocol)** is a reliable connection-oriented protocol that allows a byte stream originating on one machine to be delivered without error on any other machine in the internet. It fragments the incoming byte stream into discrete messages and passes each one onto the internet layer. At the destination, the receiving TCP process reassembles the received messages into the output stream. TCP also handles flow control to make sure a fast sender cannot swamp a slow receiver with more messages than it can handle.

The second protocol in this layer, **UDP (User Datagram Protocol)**, is an unreliable, connectionless protocol for applications that do not want TCP's sequencing or flow control and wish to provide their own. It is also widely used for one-shot, client-server type request-reply queries and applications in which prompt delivery is more important than accurate delivery, such as transmitting speech or video. The relation of IP, TCP, and UDP is shown in Fig. 1-19. Since the model was developed, IP has been implemented on many other networks.

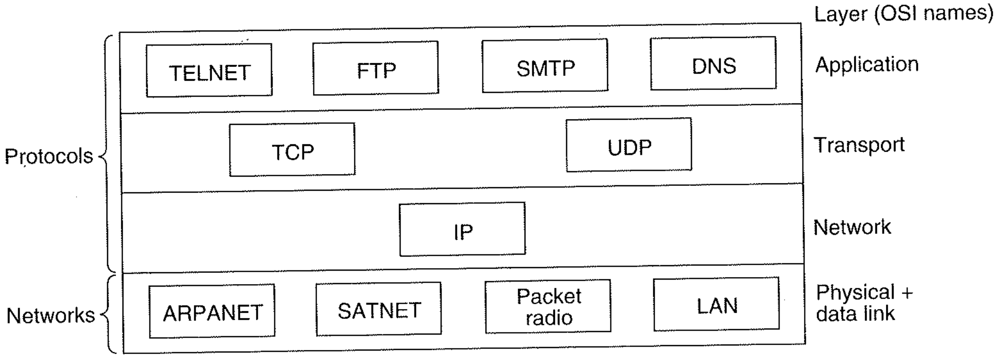


Fig. 1-19. Protocols and networks in the TCP/IP model initially.

The Application Layer

The TCP/IP model does not have session or presentation layers. No need for them was perceived, so they were not included. Experience with the OSI model has proven this view correct: they are of little use to most applications.

On top of the transport layer is the **application layer**. It contains all the higher-level protocols. The early ones included virtual terminal (TELNET), file transfer (FTP), and electronic mail (SMTP), as shown in Fig. 1-19. The virtual terminal protocol allows a user on one machine to log into a distant machine and work there. The file transfer protocol provides a way to move data efficiently from one machine to another. Electronic mail was originally just a kind of file transfer, but later a specialized protocol was developed for it. Many other protocols have been added to these over the years, such as the Domain Name Service (DNS) for mapping host names onto their network addresses, NNTP, the protocol used for moving news articles around, and HTTP, the protocol used for fetching pages on the World Wide Web, and many others.

TELNET

FTP

SMTP

The Host-to-Network Layer

Below the internet layer is a great void. The TCP/IP reference model does not really say much about what happens here, except to point out that the host has to connect to the network using some protocol so it can send IP packets over it. This protocol is not defined and varies from host to host and network to network. Books and papers about the TCP/IP model rarely discuss it.

1.4.3. A Comparison of the OSI and TCP Reference Models

The OSI and TCP/IP reference models have much in common. Both are based on the concept of a stack of independent protocols. Also, the functionality of the layers is roughly similar. For example, in both models the layers up through and including the transport layer are there to provide an end-to-end network-independent transport service to processes wishing to communicate. These layers form the transport provider. Again in both models, the layers above transport are application-oriented users of the transport service.

Despite these fundamental similarities, the two models also have many differences. In this section we will focus on the key differences between the two reference models. It is important to note that we are comparing the *reference models* here, not the corresponding *protocol stacks*. The protocols themselves will be discussed later. For an entire book comparing and contrasting TCP/IP and OSI, see (Piscitello and Chapin, 1993).

Three concepts are central to the OSI model:

1. Services
2. Interfaces
3. Protocols

Probably the biggest contribution of the OSI model is to make the distinction between these three concepts explicit. Each layer performs some services for the layer above it. The *service* definition tells what the layer does, not how entities above it access it or how the layer works.

A layer's *interface* tells the processes above it how to access it. It specifies what the parameters are and what results to expect. It, too, says nothing about how the layer works inside.

Finally, the peer *protocols* used in a layer are the layer's own business. It can use any protocols it wants to, as long as it gets the job done (i.e., provides the offered services). It can also change them at will without affecting software in higher layers.

These ideas fit very nicely with modern ideas about object-oriented programming. An object, like a layer, has a set of methods (operations) that processes outside the object can invoke. The semantics of these methods define the set of services that the object offers. The methods' parameters and results form the object's interface. The code internal to the object is its protocol and is not visible or of any concern outside the object.

The TCP/IP model did not originally clearly distinguish between service, interface, and protocol, although people have tried to retrofit it after the fact to make it more OSI-like. For example, the only real services offered by the internet layer are SEND IP PACKET and RECEIVE IP PACKET.

As a consequence, the protocols in the OSI model are better hidden than in the TCP/IP model and can be replaced relatively easily as the technology changes. Being able to make such changes is one of the main purposes of having layered protocols in the first place.

The OSI reference model was devised *before* the protocols were invented. This ordering means that the model was not biased toward one particular set of protocols, which made it quite general. The down side of this ordering is that the designers did not have much experience with the subject and did not have a good idea of which functionality to put in which layer.

For example, the data link layer originally dealt only with point-to-point networks. When broadcast networks came around, a new sublayer had to be hacked into the model. When people started to build real networks using the OSI model and existing protocols, it was discovered that they did not match the required service specifications (wonder of wonders), so convergence sublayers had to be grafted onto the model to provide a place for papering over the differences. Finally, the committee originally expected that each country would have one network, run by the government and using the OSI protocols, so no thought was given to internetworking. To make a long story short, things did not turn out that way.

With the TCP/IP the reverse was true: the protocols came first, and the model was really just a description of the existing protocols. There was no problem with the protocols fitting the model. They fit perfectly. The only trouble was that the *model* did not fit any other protocol stacks. Consequently, it was not especially useful for describing other non-TCP/IP networks.

Turning from philosophical matters to more specific ones, an obvious difference between the two models is the number of layers: the OSI model has seven layers and the TCP/IP has four layers. Both have (inter)network, transport, and application layers, but the other layers are different.

Another difference is in the area of connectionless versus connection-oriented communication. The OSI model supports both connectionless and connection-oriented communication in the network layer, but only connection-oriented communication in the transport layer, where it counts (because the transport service is visible to the users). The TCP/IP model has only one mode in the network layer (connectionless) but supports both modes in the transport layer, giving the users a choice. This choice is especially important for simple request-response protocols.

1.4.4. A Critique of the OSI Model and Protocols

Neither the OSI model and its protocols nor the TCP/IP model and its protocols are perfect. Quite a bit of criticism can be, and has been, directed at both of them. In this section and the next one, we will look at some of these criticisms. We will begin with OSI and examine TCP/IP afterward.

At the time the second edition of this book was published (1989), it appeared to most experts in the field that the OSI model and its protocols were going to take over the world and push everything else out of their way. This did not happen. Why? A look back at some of the lessons may be useful. These lessons can be summarized as:

1. Bad timing.
2. Bad technology.
3. Bad implementations.
4. Bad politics.

Bad Timing

First let us look at reason one: bad timing. The time at which a standard is established is absolutely critical to its success. David Clark of M.I.T. has a theory of standards that he calls the *apocalypse of the two elephants*, and which is illustrated in Fig. 1-20.

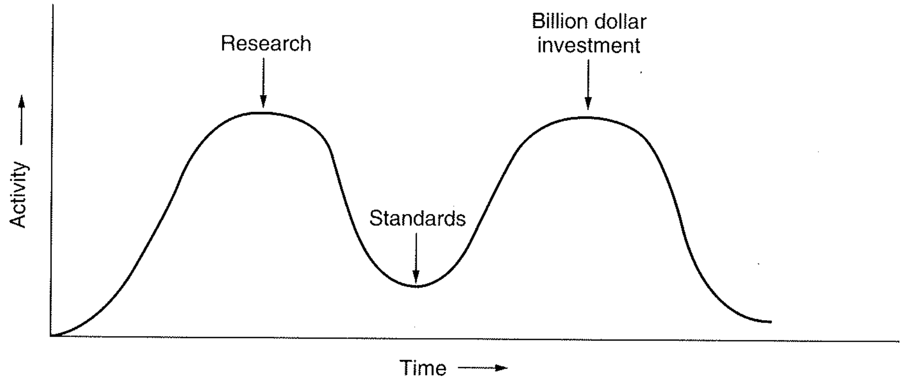


Fig. 1-20. The apocalypse of the two elephants.

This figure shows the amount of activity surrounding a new subject. When the subject is first discovered, there is a burst of research activity in the form of discussions, papers, and meetings. After a while this subsides, corporations discover the subject, and the billion-dollar wave of investment hits.

It is essential that the standards be written in the trough between the two "elephants." If they are written too early, before the research is finished, the subject may still be poorly understood, which leads to bad standards. If they are written too late, so many companies may have already made major investments in different ways of doing things that the standards are effectively ignored. If the interval between the two elephants is very short (because everyone is in a hurry to get started), the people developing the standards may get crushed.

It now appears that the standard OSI protocols got crushed. The competing TCP/IP protocols were already in widespread use by research universities by the time the OSI protocols appeared. While the billion-dollar wave of investment had not yet hit, the academic market was large enough that many vendors had begun cautiously offering TCP/IP products. When OSI came around, they did not want to support a second protocol stack until they were forced to, so there were no initial offerings. With every company waiting for every other company to go first, no company went first and OSI never happened.

Bad Technology

The second reason that OSI never caught on is that both the model and the protocols are flawed. Most discussions of the seven-layer model give the impression that the number and contents of the layers eventually chosen were the only way, or at least the obvious way. This is far from true. The session layer has little use in most applications, and the presentation layer is nearly empty. In fact, the British proposal to ISO only had five layers, not seven. In contrast to the session and presentation layers, the data link and network layers are so full that subsequent work has split them into multiple sublayers, each with different functions.

Although hardly anyone ever admits it in public, the real reason that the OSI model has seven layers is that at the time it was designed, IBM had a proprietary seven-layer protocol called **SNA**™ (**Systems Network Architecture**). At that time, IBM so dominated the computer industry that everyone else, including telephone companies, competing computer companies, and even major governments, were scared to death that IBM would use its market clout to effectively force everybody to use SNA, which it could change whenever it wished. The idea behind OSI was to produce an IBM-like reference model and protocol stack that would become the world standard, and controlled not by one company, but by a neutral organization, ISO.

The OSI model, along with the associated service definitions and protocols, is extraordinarily complex. When piled up, the printed standards occupy a significant fraction of a meter of paper. They are also difficult to implement and inefficient in operation. In this context, a riddle posed by Paul Mockapetris and cited in (Rose, 1993) comes to mind:

Q: What do you get when you cross a mobster with an international standard?
A: Someone who makes you an offer you can't understand.

In addition to being incomprehensible, another problem with OSI is that some functions, such as addressing, flow control, and error control reappear again and again in each layer. Saltzer et al. (1984), for example, have pointed out that to be effective, error control must be done in the highest layer, so that repeating it over and over in each of the lower layers is often unnecessary and inefficient.

Another issue is that the decision to place certain features in particular layers is not always obvious. The virtual terminal handling (now in the application layer) was in the presentation layer during much of the development of the standard. It was moved to the application layer because the committee had trouble deciding what the presentation layer was good for. Data security and encryption were so controversial that no one could agree which layer to put them in, so they were left out altogether. Network management was also omitted from the model for similar reasons.

Another criticism of the original standard is that it completely ignored connectionless services and connectionless protocols, even though most local area networks work that way. Subsequent addenda (known in the software world as bug fixes) corrected this problem.

Perhaps the most serious criticism is that the model is dominated by a communications mentality. The relationship of computing to communications is barely mentioned anywhere, and some of the choices made are wholly inappropriate to the way computers and software work. As an example, consider the OSI primitives, listed in Fig. 1-14. In particular, think carefully about the primitives and how one might use them in a programming language.

The CONNECT.request primitive is simple. One can imagine a library procedure, *connect*, that programs can call to establish a connection. Now think about CONNECT.indication. When a message arrives, the destination process has to be signaled. In effect, it has to get an interrupt—hardly an appropriate concept for programs written in any modern high-level language. Of course, in the lowest layer, an indication (interrupt) does occur.

If the program were expecting an incoming call, it could call a library procedure *receive* to block itself. But if this were the case, why was *receive* not the primitive instead of *indication*? *Receive* is clearly oriented toward the way computers work, whereas *indication* is equally clearly oriented toward the way telephones work. Computers are different from telephones. Telephones ring. Computers do not ring. In short, the semantic model of an interrupt-driven system is conceptually a poor idea and totally at odds with all modern ideas of structured programming. This and similar problems are discussed by Langsford (1984).

Bad Implementations

Given the enormous complexity of the model and the protocols, it will come as no surprise that the initial implementations were huge, unwieldy, and slow. Everyone who tried them got burned. It did not take long for people to associate

"OSI" with "poor quality." While the products got better in the course of time, the image stuck.

In contrast, one of the first implementations of TCP/IP was part of Berkeley UNIX® and was quite good (not to mention, free). People began using it quickly, which led to a large user community, which led to improvements, which led to an even larger community. Here the spiral was upward instead of downward.

Bad Politics

On account of the initial implementation, many people, especially in academia, thought of TCP/IP as part of UNIX, and UNIX in the 1980s in academia was not unlike parenthood (then incorrectly called motherhood) and apple pie.

OSI, on the other hand, was thought to be the creature of the European telecommunication ministries, the European Community, and later the U.S. Government. This belief was only partly true, but the very idea of a bunch of government bureaucrats trying to shove a technically inferior standard down the throats of the poor researchers and programmers down in the trenches actually developing computer networks did not help much. Some people viewed this development in the same light as IBM announcing in the 1960s that PL/I was the language of the future, or DoD correcting this later by announcing that it was actually Ada®.

Despite the fact that the OSI model and protocols have been less than a resounding success, there are still a few organizations interested in it, mostly European PTTs that still have a monopoly on telecommunication. Consequently a feeble effort has been made to update OSI, resulting in a revised model published in 1994. For what was changed (little) and what should have been changed (a lot), see (Day, 1995).

1.4.5. A Critique of the TCP/IP Reference Model

The TCP/IP model and protocols have their problems too. First, the model does not clearly distinguish the concepts of service, interface, and protocol. Good software engineering practice requires differentiating between the specification and the implementation, something that OSI does very carefully, and TCP/IP does not. Consequently, the TCP/IP model is not much of a guide for designing new networks using new technologies.

Second, the TCP/IP model is not at all general and is poorly suited to describing any protocol stack other than TCP/IP. Trying to describe SNA using the TCP/IP model would be nearly impossible, for example.

Third, the host-to-network layer is not really a layer at all in the normal sense that the term is used in the context of layered protocols. It is an interface (between the network and data link layers). The distinction between an interface and a layer is a crucial one and one should not be sloppy about it.

Fourth, the TCP/IP model does not distinguish (or even mention) the physical and data link layers. These are completely different. The physical layer has to do with the transmission characteristics of copper wire, fiber optics, and wireless communication. The data link layer's job is to delimit the start and end of frames and get them from one side to the other with the desired degree of reliability. A proper model should include both as separate layers. The TCP/IP model does not do this.

Finally, although the IP and TCP protocols were carefully thought out, and well implemented, many of the other protocols were ad hoc, generally produced by a couple of graduate students hacking away until they got tired. The protocol implementations were then distributed free, which resulted in their becoming widely used, deeply entrenched, and thus hard to replace. Some of them are a bit of an embarrassment now. The virtual terminal protocol, TELNET, for example, was designed for a ten-character per second mechanical Teletype terminal. It knows nothing of graphical user interfaces and mice. Nevertheless, 25 years later, it is still in widespread use.

In summary, despite its problems, the OSI *model* (minus the session and presentation layers) has proven to be exceptionally useful for discussing computer networks. In contrast, the OSI *protocols* have not become popular. The reverse is true of TCP/IP: the *model* is practically nonexistent, but the *protocols* are widely used. Since computer scientists like to have their cake and eat it, too, in this book we will use a modified OSI model but concentrate primarily on the TCP/IP and related protocols, as well as newer ones such as SMDS, frame relay, SONET, and ATM. In effect, we will use the hybrid model of Fig. 1-21 as the framework for this book.

5	Application layer
4	Transport layer
3	Network layer
2	Data Link layer
1	Physical layer

Fig. 1-21. The hybrid reference model to be used in this book.

1.5. EXAMPLE NETWORKS

Numerous networks are currently operating around the world. Some of these are public networks run by common carriers or PTTs, others are research networks, yet others are cooperative networks run by their users, and still others are commercial or corporate networks. In the following sections we will take a look usage must be below a predetermined level. In return, the carrier charges much less for a virtual line than a physical one.

In addition to competing with leased lines, frame relay also competes with X.25 permanent virtual circuits, except that it operates at higher speeds, usually 1.5 Mbps, and provides fewer features.

Frame relay provides a minimal service, primarily a way to determine the start and end of each frame, and detection of transmission errors. If a bad frame is received, the frame relay service simply discards it. It is up to the user to discover that a frame is missing and take the necessary action to recover. Unlike X.25, frame relay does not provide acknowledgements or normal flow control. It does have a bit in the header, however, which one end of a connection can set to indicate to the other end that problems exist. The use of this bit is up to the users.

1.6.4. Broadband ISDN and ATM

Even if the above services become popular, the telephone companies are still faced with a far more fundamental problem: multiple networks. POTS (Plain Old Telephone Service) and Telex use the old circuit-switched network. Each of the new data services such as SMDS and frame relay uses its own packet-switching network. DQDB is different from these, and the internal telephone company call management network (SSN 7) is yet another network. Maintaining all these separate networks is a major headache, and there is another network, cable television, that the telephone companies do not control and would like to.

The perceived solution is to invent a single new network for the future that will replace the entire telephone system and all the specialized networks with a single integrated network for all kinds of information transfer. This new network will have a huge data rate compared to all existing networks and services and will make it possible to offer a large variety of new services. This is not a small project, and it is certainly not going to happen overnight, but it is now under way.

The new wide area service is called **B-ISDN** (**Broadband Integrated Services Digital Network**). It will offer video on demand, live television from many sources, full motion multimedia electronic mail, CD-quality music, LAN interconnection, high-speed data transport for science and industry and many other services that have not yet even been thought of, all over the telephone line.

The underlying technology that makes B-ISDN possible is called **ATM** (**Asynchronous Transfer Mode**) because it is not synchronous (tied to a master clock), as most long distance telephone lines are. Note that the acronym ATM here has nothing to do with the Automated Teller Machines many banks provide (although an ATM machine can use an ATM network to talk to its bank).

A great deal of work has already been done on ATM and on the B-ISDN system that uses it, although there is more ahead. For more information on this subject, see (Fischer et al., 1994; Gasman, 1994; Goralski, 1995; Kim et al., 1994; Kyas, 1995; McDysan and Spohn, 1995; and Stallings, 1995a).

The basic idea behind ATM is to transmit all information in small, fixed-size packets called **cells**. The cells are 53 bytes long, of which 5 bytes are header and 48 bytes are payload, as shown in Fig. 1-29. ATM is both a technology (hidden from the users) and potentially a service (visible to the users). Sometimes the service is called **cell relay**, as an analogy to frame relay.

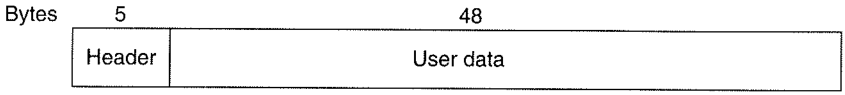


Fig. 1-29. An ATM cell.

The use of a cell-switching technology is a gigantic break with the 100-year old tradition of circuit switching (establishing a copper path) within the telephone system. There are a variety of reasons why cell switching was chosen, among them are the following. First, cell switching is highly flexible and can handle both constant rate traffic (audio, video) and variable rate traffic (data) easily. Second, at the very high speeds envisioned (gigabits per second are within reach), digital switching of cells is easier than using traditional multiplexing techniques, especially using fiber optics. Third, for television distribution, broadcasting is essential; cell switching can provide this and circuit switching cannot.

ATM networks are connection-oriented. Making a call requires first sending a message to set up the connection. After that, subsequent cells all follow the same path to the destination. Cell delivery is not guaranteed, but their order is. If cells 1 and 2 are sent in that order, then if both arrive, they will arrive in that order, never first 2 then 1.

ATM networks are organized like traditional WANs, with lines and switches (routers). The intended speeds for ATM networks are 155 Mbps and 622 Mbps, with the possibility of gigabit speeds later. The 155-Mbps speed was chosen because this is about what is needed to transmit high definition television. The exact choice of 155.52 Mbps was made for compatibility with AT&T's SONET transmission system. The 622 Mbps speed was chosen so four 155-Mbps channels could be sent over it. By now it should be clear why some of the gigabit testbeds operated at 622 Mbps: they used ATM.

When ATM was proposed, virtually all the discussion (i.e., the hype) was about video on demand to every home and replacing the telephone system, as described above. Since then, other developments have become important. Many organizations have run out of bandwidth on their campus or building-wide LANs and are being forced to go to some kind of switched system that has more bandwidth than does a single LAN. Also, in client-server computing, some applications need the ability to talk to certain servers at high speed. ATM is certainly a major candidate for both of these applications. Nevertheless, it is a bit of a letdown to go from a goal of trying to replace the entire low-speed analog telephone system with a high-speed digital one to a goal of trying connect all the Ethernets on campus. LAN interconnection using ATM is discussed in (Kavak, 1995; Newman, 1994; and Truong et al., 1995).

It is also worth pointing out that different organizations involved in ATM have different (financial) interests. The long-distance telephone carriers and PTTs are mostly interested in using ATM to upgrade the telephone system and compete with the cable TV companies in electronic video distribution. The computer vendors see campus ATM LANs as the big moneymaker (for them). All these competing interests do not make the ongoing standardization process any easier, faster, or more coherent. Also, politics and power within the organization standardizing ATM (The ATM Forum) have considerable influence on where ATM is going.

The B-ISDN ATM Reference Model

Let us now turn back to the technology of ATM, especially as used in the (future) telephone system. Broadband ISDN using ATM has its own reference model, different from the OSI model and also different from the TCP/IP model. This model is shown in Fig. 1-30. It consists of three layers, the physical, ATM, and ATM adaptation layers, plus whatever the users want to put on top of that.

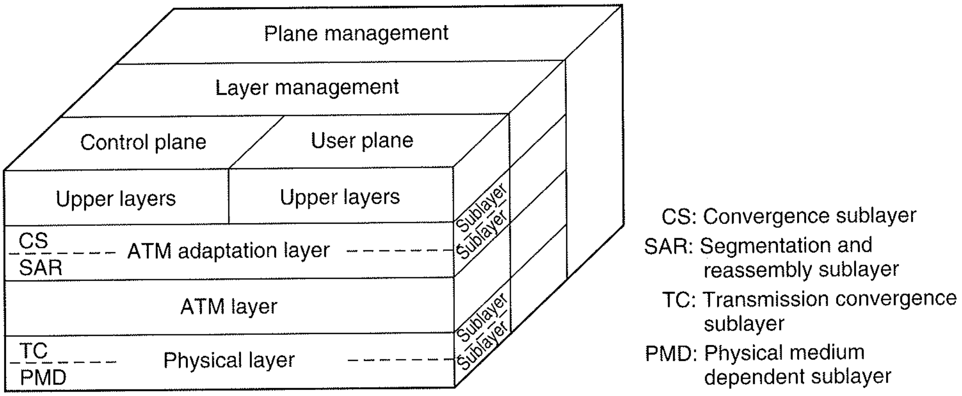


Fig. 1-30. The B-ISDN ATM reference model.

The physical layer deals with the physical medium: voltages, bit timing, and various other issues. ATM does not prescribe a particular set of rules, but instead says that ATM cells may be sent on a wire or fiber by themselves, but they may also be packaged inside the payload of other carrier systems. In other words, ATM has been designed to be independent of the transmission medium.

The **ATM layer** deals with cells and cell transport. It defines the layout of a cell and tells what the header fields mean. It also deals with establishment and release of virtual circuits. Congestion control is also located here.

Because most applications do not want to work directly with cells (although some may), a layer above the ATM layer has been defined that allows users to send packets larger than a cell. The ATM interface segments these packets, transmits the cells individually, and reassembles them at the other end. This layer is the **AAL (ATM Adaptation Layer)**.

Unlike the earlier two-dimensional reference models, the ATM model is defined as being three-dimensional, as shown in Fig. 1-30. The **user plane** deals with data transport, flow control, error correction, and other user functions. In contrast, the **control plane** is concerned with connection management. The layer and plane management functions relate to resource management and interlayer coordination.

The physical and AAL layers are each divided into two sublayers, one at the bottom that does the work and a convergence sublayer on top that provides the proper interface to the layer above it. The functions of the layers and sublayers are given in Fig. 1-31.

OSI layer	ATM layer	ATM sublayer	Functionality
3/4	AAL	CS	Providing the standard interface (convergence)
		SAR	Segmentation and reassembly
2/3	ATM		Flow control Cell header generation/extraction Virtual circuit/path management Cell multiplexing/demultiplexing
2	Physical	TC	Cell rate decoupling Header checksum generation and verification Cell generation Packing/unpacking cells from the enclosing envelope Frame generation
1		PMD	Bit timing Physical network access

Fig. 1-31. The ATM layers and sublayers, and their functions.

The **PMD (Physical Medium Dependent)** sublayer interfaces to the actual cable. It moves the bits on and off and handles the bit timing. For different carriers and cables, this layer will be different.

The other sublayer of the physical layer is the **TC (Transmission Convergence)** sublayer. When cells are transmitted, the TC layer sends them as a string of bits to the PMD layer. Doing this is easy. At the other end, the TC sublayer gets a pure incoming bit stream from the PMD sublayer. Its job is to convert this bit stream into a cell stream for the ATM layer. It handles all the issues related to telling where cells begin and end in the bit stream. In the ATM model, this functionality is in the physical layer. In the OSI model and in pretty much all other networks, the job of framing, that is, turning a raw bit stream into a sequence of frames or cells, is the data link layer's task. For that reason we will discuss it in this book along with the data link layer, not with the physical layer.

As we mentioned earlier, the ATM layer manages cells, including their generation and transport. Most of the interesting aspects of ATM are located here. It is a mixture of the OSI data link and network layers, but it is not split into sublayers.

The AAL layer is split into a **SAR** (**Segmentation And Reassembly**) sublayer and a **CS** (**Convergence Sublayer**). The lower sublayer breaks packets up into cells on the transmission side and puts them back together again at the destination. The upper sublayer makes it possible to have ATM systems offer different kinds of services to different applications (e.g., file transfer and video on demand have different requirements concerning error handling, timing, etc.).

Perspective on ATM

To a considerable extent, ATM is a project invented by the telephone industry because after Ethernet was widely installed, the computer industry never rallied around any higher-speed network technology to make it standard. The telephone companies filled this vacuum with ATM, although in October 1991, many computer vendors joined with the telephone companies to set up the **ATM Forum**, an industry group that will guide the future of ATM.

Although ATM promises the ability to deliver information anywhere at speeds soon to exceed 1 Gbps, delivering on this promise will not be easy. ATM is basically high-speed packet-switching, a technology the telephone companies have little experience with. What they do have, is a massive investment in a different technology (circuit switching) that is in concept unchanged since the days of Alexander Graham Bell. Needless to say, this transition will not happen quickly, all the more so because it is a revolutionary change rather than an evolutionary one, and revolutions never go smoothly.

The economics of installing ATM worldwide also have to be considered. A substantial fraction of the existing telephone system will have to be replaced. Who will pay for this? How much will consumers be willing to pay to get a movie on demand electronically, when they can get one at the local video store for a couple of dollars? Finally, the question of where many of the advanced services are provided is crucial. If they are provided by the network, the telephone companies will profit from them. If they are provided by computers attached to the network, the manufacturers and operators of these devices make the profits. The users may not care, but the telephone companies and computer vendors certainly do, and this will surely affect their interest in making ATM happen.

1.6.5. Comparison of Services

The reader may be wondering why so many incompatible and overlapping services exist, including DQDB, SMDS, X.25, frame relay, ATM, and more. The underlying reason is the 1984 decision to break up AT&T and foster competition in the telecommunications industry. Different companies with different interests and technologies are now free to offer whatever services they think there is a demand for, and many of them are doing this with a vengeance.

To recap some of the services we have touched on in this chapter, DQDB is an unswitched MAN technology that allows 53-byte cells (of which 44 are payload) to be sent down long wires within a city. SMDS is a switched datagram technology for sending datagrams anywhere in a network at 45 Mbps. X.25 is an older connection-oriented networking technology for transmitting small variable-sized packets at 64 kbps. Frame relay is a service that provides virtual leased lines at speeds around 1.5 Mbps. Finally, ATM is designed to replace the entire circuit-switched telephone system with cell switching and be able to handle data and television as well. Some differences between these competitors are summarized in Fig. 1-32.

Issue	DQDB	SMDS	X.25	Frame Relay	ATM AAL
Connection oriented	Yes	No	Yes	Yes	Yes
Normal speed (Mbps)	45	45	.064	1.5	155
Switched	No	Yes	Yes	No	Yes
Fixed-size payload	Yes	No	No	No	No
Max payload	44	9188	128	1600	Variable
Permanent VCs	No	No	Yes	Yes	Yes
Multicasting	No	Yes	No	No	Yes

Fig. 1-32. Different networking services.

1.7. NETWORK STANDARDIZATION

Many network vendors and suppliers exist, each with their own ideas of how things should be done. Without coordination, there would be complete chaos, and users would be able to get nothing done. The only way out is to agree upon some network standards.

Not only do standards allow different computers to communicate, but they also increase the market for products adhering to the standard, which leads to

2.4. THE TELEPHONE SYSTEM

When two computers owned by the same company or organization and located close to each other need to communicate, it is often easiest just to run a cable between them. LANs work this way. However, when the distances are large, or there are many computers, or the cables would have to pass through a public road or other public right of way, the costs of running private cables are usually prohibitive. Furthermore, in just about every country in the world, stringing private transmission lines across (or underneath) public property is also illegal. Consequently, the network designers must rely upon the existing telecommunication facilities.

These facilities, especially the **PSTN**, (**Public Switched Telephone Network**), were usually designed many years ago, with a completely different goal in mind: transmitting the human voice in a more or less recognizable form. Their suitability for use in computer-computer communication is often marginal at best, but the situation is rapidly changing with the introduction of fiber optics and digital technology. In any event, the telephone system is so tightly intertwined with (wide area) computer networks, that it is worth devoting considerable time studying it.

To see the order of magnitude of the problem, let us make a rough but illustrative comparison of the properties of a typical computer-computer connection via a local cable and via a dial-up telephone line. A cable running between two computers can transfer data at memory speeds, typically 10^7 to 10^8 bps. The error rate is usually so low that it is hard to measure, but one error per day would be considered poor at most installations. One error per day at these speeds is equivalent to one error per 10^{12} or 10^{13} bits sent.

In contrast, a dial-up line has a maximum data rate on the order of 10^4 bps and an error rate of roughly 1 per 10^5 bits sent, varying somewhat with the age of the telephone switching equipment involved. The combined bit rate times error rate performance of a local cable is thus 11 orders of magnitude better than a voice-grade telephone line. To make an analogy in the field of transportation, the ratio of the cost of the entire Apollo project, which landed men on the moon, to the cost of a bus ride downtown is about 11 orders of magnitude (in 1965 dollars: 40 billion to 0.40).

The trouble, of course, is that computer systems designers are used to working with computer systems, and when suddenly confronted with another system whose performance (from their point of view) is 11 orders of magnitude worse, it is not surprising that much time and effort have been devoted to trying to figure out how to use it efficiently. On the other hand, the telephone companies have made massive strides in the past decade in upgrading equipment and improving service in certain areas. In the following sections we will describe the telephone system and show what it used to be and where it is going. For additional information about the innards of the telephone system see (Bellamy, 1991).

2.4.1. Structure of the Telephone System

When Alexander Graham Bell patented the telephone in 1876 (just a few hours ahead of his rival, Elisha Gray), there was an enormous demand for his new invention. The initial market was for the sale of telephones, which came in pairs. It was up to the customer to string a single wire between them. The electrons returned through the earth. If a telephone owner wanted to talk to *n* other telephone owners, separate wires had to be strung to all *n* houses. Within a year, the cities were covered with wires passing over houses and trees in a wild jumble. It became immediately obvious that the model of connecting every telephone to every other telephone, as shown in Fig. 2-14(a) was not going to work.

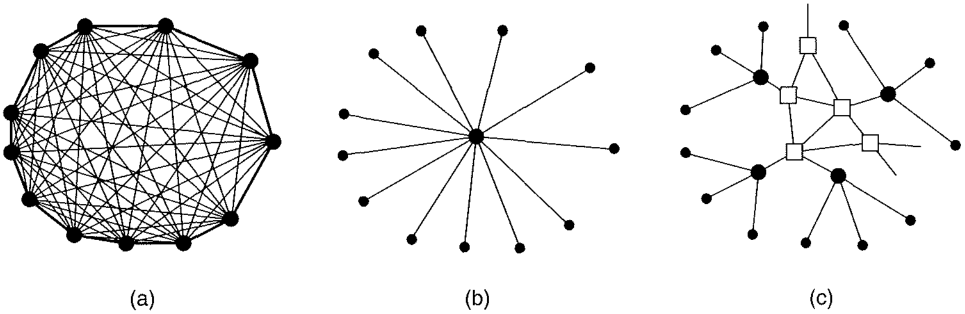


Fig. 2-14. (a) Fully interconnected network. (b) Centralized switch. (c) Two-level hierarchy.

To his credit, Bell saw this and formed the Bell Telephone Company, which opened its first switching office (in New Haven, Connecticut) in 1878. The company ran a wire to each customer's house or office. To make a call, the customer would crank the phone to make a ringing sound in the telephone company office to attract the attention of an operator, who would then manually connect the caller to the callee using a jumper cable. The model of a single switching office is illustrated in Fig. 2-14(b).

Pretty soon, Bell System switching offices were springing up everywhere and people wanted to make long-distance calls between cities, so the Bell system began to connect the switching offices. The original problem soon returned: to connect every switching office to every other switching office by means of a wire between them quickly became unmanageable, so second-level switching offices were invented, After a while, multiple second-level offices were needed, as shown in Fig. 2-14(c). Eventually, the hierarchy grew to five levels.

By 1890, the three major parts of the telephone system were in place: the switching offices, the wires between the customers and the switching offices (by now balanced, insulated, twisted pairs instead of open wires with an earth return), and the long-distance connections between the switching offices. While there have been improvements in all three areas since then, the basic Bell System model has remained essentially intact for over 100 years. For a short technical history of the telephone system, see (Hawley, 1991).

At present, the telephone system is organized as a highly redundant, multilevel hierarchy. The following description is highly simplified but gives the essential flavor nevertheless. Each telephone has two copper wires coming out of it that go directly to the telephone company's nearest **end office** (also called a **local central office**). The distance is typically 1 to 10 km, being smaller in cities than in rural areas.

In the United States alone there are about 19,000 end offices. The concatenation of the area code and the first three digits of the telephone number uniquely specify an end office, which is why the rate structure uses this information. The two-wire connections between each subscriber's telephone and the end office are known in the trade as the **local loop**. If the world's local loops were stretched out end to end, they would extend to the moon and back 1000 times.

At one time, 80 percent of AT&T's capital value was the copper in the local loops. AT&T was then, in effect, the world's largest copper mine. Fortunately, this fact was not widely known in the investment community. Had it been known, some corporate raider might have bought AT&T, terminated all telephone service in the United States, ripped out all the wire, and sold the wire to a copper refiner to get a quick payback.

If a subscriber attached to a given end office calls another subscriber attached to the same end office, the switching mechanism within the office sets up a direct electrical connection between the two local loops. This connection remains intact for the duration of the call.

If the called telephone is attached to another end office, a different procedure has to be used. Each end office has a number of outgoing lines to one or more nearby switching centers, called **toll offices** (or if they are within the same local area, **tandem offices**). These lines are called **toll connecting trunks**. If both the caller's and callee's end offices happen to have a toll connecting trunk to the same toll office (a likely occurrence if they are relatively close by), the connection may be established within the toll office. A telephone network consisting only of telephones (the small dots), end offices (the large dots) and toll offices (the squares) is shown in Fig. 2-14(c).

If the caller and callee do not have a toll office in common, the path will have to be established somewhere higher up in the hierarchy. There are primary, sectional, and regional offices that form a network by which the toll offices are connected. The toll, primary, sectional, and regional exchanges communicate with each other via high bandwidth **intertoll trunks** (also called **interoffice trunks**). The number of different kinds of switching centers and their topology (e.g., may two sectional offices have a direct connection or must they go through a regional office?) varies from country to country depending on its telephone density. Figure 2-15 shows how a medium-distance connection might be routed.

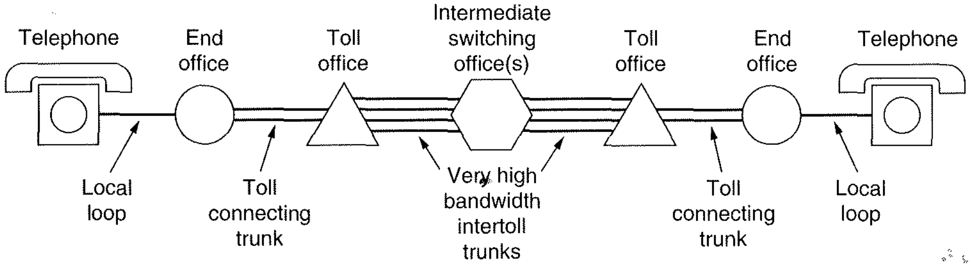


Fig. 2-15. Typical circuit route for a medium-distance call.

A variety of transmission media are used for telecommunication. Local loops consist of twisted pairs nowadays, although in the early days of telephony, uninsulated wires spaced 25 cm apart on telephone poles were common. Between switching offices, coaxial cables, microwaves, and especially fiber optics are widely used.

In the past, signaling throughout the telephone system was analog, with the actual voice signal being transmitted as an electrical voltage from source to destination. With the advent of digital electronics and computers, digital signaling has become possible. In this system, only two voltages are allowed, for example –5 volts and +5 volts.

This scheme has a number of advantages over analog signaling. First is that although the attenuation and distortion are more severe when sending two-level signals than when using modems, it is easy to calculate how far a signal can propagate and still be recognizable. A digital regenerator can be inserted into the line there, to restore the signal to its original value, since there are only two possibilities. A digital signal can pass through an arbitrary number of regenerators with no loss in signal and thus travel long distances with no information loss. In contrast, analog signals always suffer some information loss when amplified, and this loss is cumulative. The net result is that digital transmission can be made to have a low error rate.

A second advantage of digital transmission is that voice, data, music, and images (e.g., television, fax, and video) can be interspersed to make more efficient use of the circuits and equipment. Another advantage is that much higher data rates are possible using existing lines.

A third advantage is that digital transmission is much cheaper than analog transmission, since it is not necessary to accurately reproduce an analog waveform after it has passed through potentially hundreds of amplifiers on a transcontinental call. Being able to correctly distinguish a 0 from a 1 is enough.

Finally, maintenance of a digital system is easier than maintenance of an analog one. A transmitted bit is either received correctly or not, making it simpler to track down problems.

Consequently, all the long-distance trunks within the telephone system are rapidly being converted to digital. The old system used analog transmission over copper wires; the new one uses digital transmission over optical fibers.

In summary, the telephone system consists of three major components:

1. Local loops (twisted pairs, analog signaling).
2. Trunks (fiber optics or microwave, mostly digital).
3. Switching offices.

After a short digression on the politics of telephones, we will come back to each of these three components in some detail. For the local loop, we will be concerned with how to send digital data over it (quick answer: use a modem). For the long-haul trunks, the main issue is how to collect multiple calls together and send them together. This subject is called multiplexing, and we will study three different ways to do it. Finally, there are two fundamentally different ways of doing switching, so we will look at both of these.

2.4.2. The Politics of Telephones

For decades prior to 1984, the Bell System provided both local and long distance service throughout most of the United States. In the 1970s, the U.S. government came to believe that this was an illegal monopoly and sued to break it up. The government won, and on Jan. 1, 1984, AT&T was broken up into AT&T Long Lines, 23 **BOCs** (**Bell Operating Companies**), and a few other pieces. The 23 BOCs were grouped together into seven regional BOCs (RBOCs) to make them economically viable. The entire nature of telecommunication in the United States was changed overnight by court order (*not* by an act of Congress).

The exact details of the divestiture were described in the so-called **MFJ** (**Modified Final Judgment**), an oxymoron if ever there was one (if the judgment could be modified, it clearly was not final). This event led to increased competition, better service, and lower prices to consumers and businesses. Many other countries are now considering introducing competition along similar lines.

To make it clear who could do what, the United States was divided up into about 160 **LATA**s (**Local Access and Transport Areas**). Very roughly, a LATA is about as big as the area covered by one area code. Within a LATA, there is normally one **LEC** (**Local Exchange Carrier**) that has a monopoly on traditional telephone service within the LATA. The most important LECs are the BOCs, although some LATAs contain one or more of the 1500 independent telephone companies operating as LECs. In geographically large LATAs (mostly in the West), the LEC may handle long distance calls within its own LATA but may not handle calls going to a different LATA.

All inter-LATA traffic is handled by a different kind of company, an **IXC** (**InterеXchange Carrier**). Originally, AT&T Long Lines was the only serious IXC, but now MCI and Sprint are well-established competitors in the IXC business. One of the concerns at the breakup was to ensure that all the IXCs would be treated equally in terms of line quality, tariffs, and the number of digits their customers would have to dial to use them. The way this is handled is illustrated in Fig. 2-16. Here we see three example LATAs, each with several end offices. LATAs 2 and 3 also have a small hierarchy with tandem offices (intra-LATA toll offices).

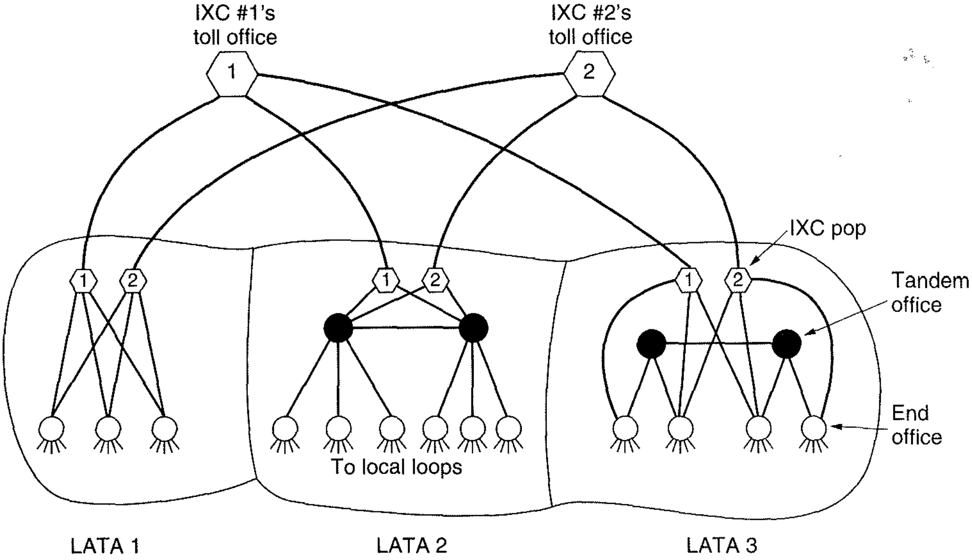


Fig. 2-16. The relationship of LATAs, LECs, and IXCs. All the circles are LEC switching offices. Each hexagon belongs to the IXC whose number is in it.

Any IXC that wishes to handle calls originating in a LATA can build a switching office called a **POP** (**Point of Presence**) there. The LEC is required to connect each IXC to every end office, either directly, as in LATAs 1 and 3, or indirectly, as in LATA 2. Furthermore, the terms of the connection, both technical and financial, must be identical for all IXCs. In this way, a subscriber in, say, LATA 1, can choose which IXC to use for calling subscribers in LATA 3.

As part of the MFJ, the IXCs were forbidden to offer local telephone service and the LECs were forbidden to offer inter-LATA telephone service, although both were free to enter other businesses, such as operating fried chicken restaurants. In 1984, that was a fairly unambiguous statement. Unfortunately, technology has a way of making the law obsolete. Neither cable television nor cellular phones were covered by the agreement. As cable television went from one way to two way, and cellular phones exploded in popularity, both LECs and IXCs began buying up or merging with cable and cellular operators.

By 1995, Congress saw that trying to maintain a distinction between the various kinds of companies was no longer tenable and drafted a bill to allow cable TV companies, local telephone companies, long distance carriers, and cellular operators to enter one another's businesses. The idea was that any company could then offer its customers a single integrated package containing cable TV, telephone, and information services, and that different companies would compete on service and price. The bill was enacted into law in February 1996. As a result, the U.S. telecommunications landscape is currently undergoing a radical restructuring.

2.4.3. The Local Loop

For the past 100 years, analog transmission has dominated all communication. In particular, the telephone system was originally based entirely on analog signaling. While the long-distance trunks are now largely digital in the more advanced countries, the local loops are still analog and are likely to remain so for at least a decade or two, due to the enormous cost of converting them. Consequently, when a computer wishes to send digital data over a dial-up line, the data must first be converted to analog form by a modem for transmission over the local loop, then converted to digital form for transmission over the long-haul trunks, then back to analog over the local loop at the receiving end, and finally back to digital by another modem for storage in the destination computer. This arrangement is shown in Fig. 2-17.

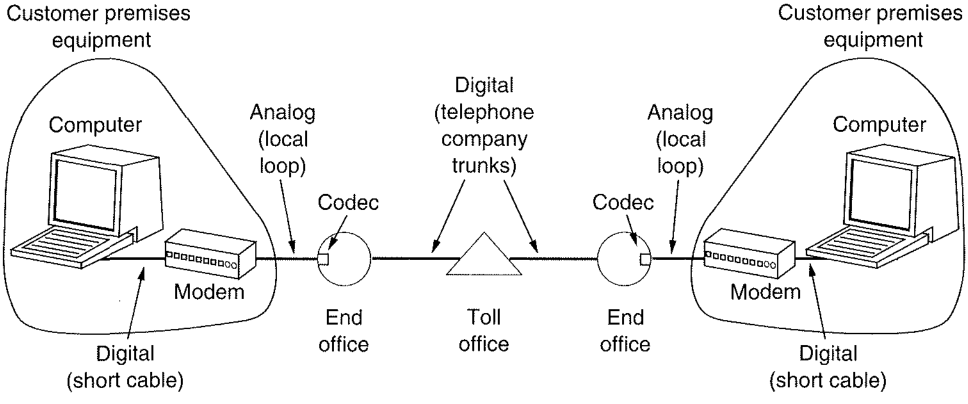


Fig. 2-17. The use of both analog and digital transmission for a computer to computer call. Conversion is done by the modems and codecs.

While this situation is not exactly ideal, such is life for the time being, and students of networking should have some understanding of both analog and digital transmission, as well as how the conversions back and forth work. For leased lines it is possible to go digital from start to finish, but these are expensive and are only useful for building intracompany private networks.

In the following sections we will look briefly at what is wrong with analog transmission and examine how modems make it possible to transmit digital data over analog circuits. We will also look at two common modem interfaces, RS-232-C and RS-449.

Transmission Impairments

Analog signaling consists of varying a voltage with time to represent an information stream. If transmission media were perfect, the receiver would receive exactly the same signal that the transmitter sent. Unfortunately, media are not perfect, so the received signal is not the same as the transmitted signal. For digital data, this difference can lead to errors.

Transmission lines suffer from three major problems: attenuation, delay distortion, and noise. **Attenuation** is the loss of energy as the signal propagates outward. On guided media (e.g., wires and optical fibers), the signal falls off logarithmically with the distance. The loss is expressed in decibels per kilometer. The amount of energy lost depends on the frequency. To see the effect of this frequency dependence, imagine a signal not as a simple waveform, but as a series of Fourier components. Each component is attenuated by a different amount, which results in a different Fourier spectrum at the receiver, and hence a different signal.

If the attenuation is too much, the receiver may not be able to detect the signal at all, or the signal may fall below the noise level. In many cases, the attenuation properties of a medium are known, so amplifiers can be put in to try to compensate for the frequency-dependent attenuation. The approach helps but can never restore the signal exactly back to its original shape.

The second transmission impairment is **delay distortion**. It is caused by the fact that different Fourier components travel at different speeds. For digital data, fast components from one bit may catch up and overtake slow components from the bit ahead, mixing the two bits and increasing the probability of incorrect reception.

The third impairment is **noise**, which is unwanted energy from sources other than the transmitter. Thermal noise is caused by the random motion of the electrons in a wire and is unavoidable. Cross talk is caused by inductive coupling between two wires that are close to each other. Sometimes when talking on the telephone, you can hear another conversation in the background. That is cross talk. Finally, there is impulse noise, caused by spikes on the power line or other causes. For digital data, impulse noise can wipe out one or more bits.

Modems

Due to the problems just discussed, especially the fact that both attenuation and propagation speed are frequency dependent, it is undesirable to have a wide range of frequencies in the signal. Unfortunately, square waves, as in digital data, have a wide spectrum and thus are subject to strong attenuation and delay distortion. These effects make baseband (DC) signaling unsuitable except at slow speeds and over short distances.

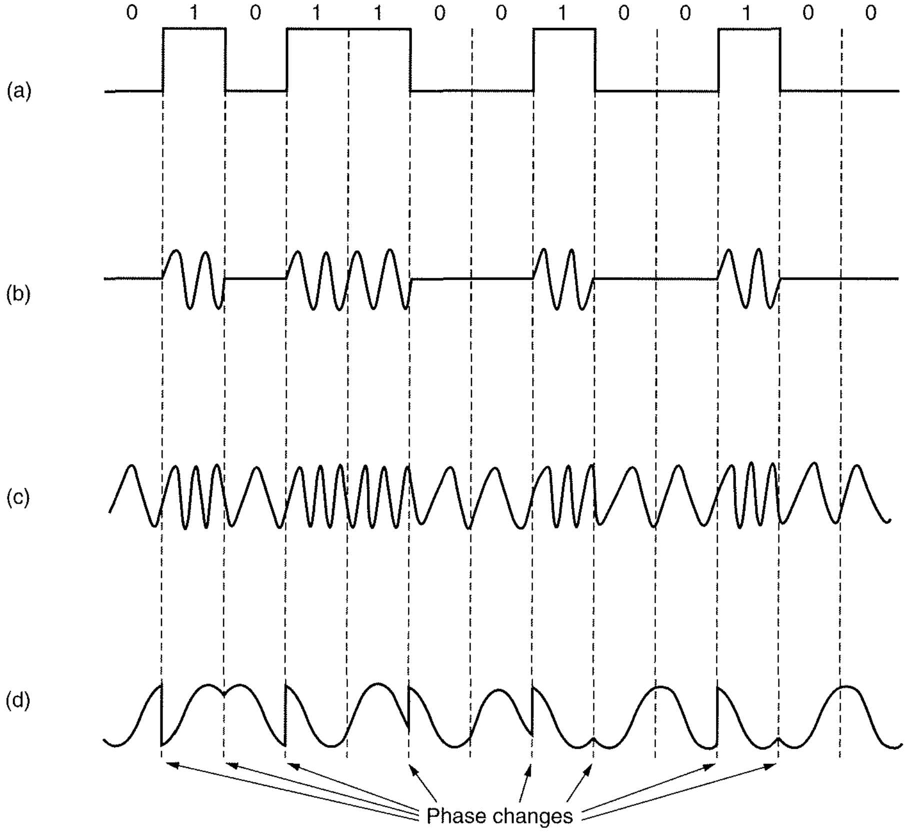


Fig. 2-18. (a) A binary signal. (b) Amplitude modulation. (c) Frequency modulation. (d) Phase modulation.

To get around the problems associated with DC signaling, especially on telephone lines, AC signaling is used. A continuous tone in the 1000- to 2000-Hz range, called a **sine wave carrier** is introduced. Its amplitude, frequency, or phase can be modulated to transmit information. In **amplitude modulation**, two different voltage levels are used to represent 0 and 1, respectively. In **frequency modulation**, also known as **frequency shift keying**, two (or more) different tones are used. In the simplest form of **phase modulation**, the carrier wave is systematically shifted 45, 135, 225, or 315 degrees at uniformly spaced intervals. Each phase shift transmits 2 bits of information. Figure 2-18 illustrates the three forms of modulation. A device that accepts a serial stream of bits as input and produces a modulated carrier as output (or vice versa) is called a **modem** (for modulator-demodulator). The modem is inserted between the (digital) computer and the (analog) telephone system.

To go to higher and higher speeds, it is not possible to just keep increasing the sampling rate. The Nyquist theorem says that even with a perfect 3000-Hz line (which a dial-up telephone is decidedly not), there is no point in sampling faster than 6000 Hz. Thus all research on faster modems is focused on getting more bits per sample (i.e., per baud).

Most advanced modems use a combination of modulation techniques to transmit multiple bits per baud. In Fig. 2-19(a), we see dots at 0, 90, 180, and 270 degrees, with two amplitude levels per phase shift. Amplitude is indicated by the distance from the origin. In Fig. 2-19(b) we see a different modulation scheme, in which 16 different combinations of amplitude and phase shift are used. Thus Fig. 2-19(a) has eight valid combinations and can be used to transmit 3 bits per baud. In contrast, Fig. 2-19(b) has 16 valid combinations and can thus be used to transmit 4 bits per baud. The scheme of Fig. 2-19(b) when used to transmit 9600 bps over a 2400-baud line is called **QAM** (**Quadrature Amplitude Modulation**).

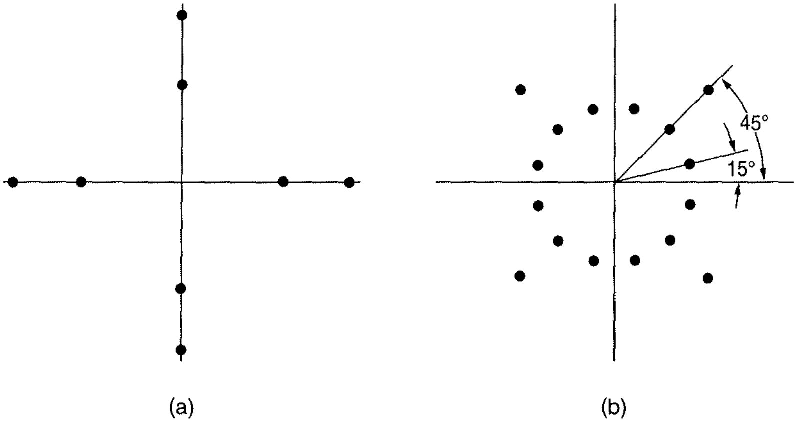


Fig. 2-19. (a) 3 bits/baud modulation. (b) 4 bits/baud modulation.

Diagrams such as those of Fig. 2-19, which show the legal combinations of amplitude and phase, are called **constellation patterns**. Each high-speed modem standard has its own constellation pattern and can talk only to other modems that use the same one (although most modems can emulate all the slower ones). The ITU V.32 9600 bps modem standard uses the constellation pattern of Fig. 2-19(b), for example.

The next step above 9600 bps is 14,400 bps. It is called **V.32 bis**. This speed is achieved by transmitting 6 bits per sample at 2400 baud. Its constellation pattern has 64 points. Fax modems use this speed to transmit pages that have been scanned in as bit maps. After V.32 bis comes **V.34**, which runs at 28,800 bps.

With so many points in the constellation pattern, even a small amout of noise in the detected amplitude or phase can result in an error, and potentially 6 bad bits. To reduce the chance of getting an error, many modems add a parity bit, giving 128 points in the constellation pattern. The coding of the points is carefully done to maximize the chance of detecting errors. The coding that does this is called **trellis coding**.

A completely different approach to high-speed transmission is to divide the available 3000-Hz spectrum into 512 tiny bands and transmit at, say, 20 bps in each one. This scheme requires a substantial processor inside the modem, but has the advantage of being able to disable frequency bands that are too noisy. Modems that use this approach normally have V.32 or V.34 capability as well, so they can talk to standard modems.

Many modems now have compression and error correction built into the modems. The big advantage of this approach is that these features improve the effective data rate without requiring any changes to existing software. One popular compression scheme is **MNP 5**, which uses run-length encoding to squeeze out runs of identical bytes. Fax modems also use run-length encoding, since runs of 0s (blank paper) are very common. Another scheme is **V.42 bis**, which uses a Ziv-Lempel compression algorithm also used in Compress and other programs (Ziv and Lempel, 1977).

Even when modems are used, another problem can occur on telephone lines: echoes. On a long line, when the signal gets to the final destination, some of the energy may be reflected back, analogous to acoustic echos in the mountains. As an illustration of electromagnetic echoes, try shining a flashlight from a darkened room through a closed window at night. You will see a reflection of the flashlight in the window (i.e., some of the energy has been reflected at the air-glass junction and sent back toward you). The same thing happens on transmission lines, especially at the point where the local loop terminates in the end office.

The effect of the echo is that a person speaking on the telephone hears his own words after a short delay. Psychological studies have shown that this is annoying to many people, often making them stutter or become confused. To eliminate the problem of echoes, echo suppressors are installed on lines longer than 2000 km. (On short lines the echoes come back so fast that people are not bothered by them.) An **echo suppressor** is a device that detects human speech coming from one end of the connection and suppresses all signals going the other way. It is basically an amplifier than can be switched on and off by a control signal produced by a speech detection circuit.

When the first person stops talking and the second begins, the echo suppressor switches directions. A good echo suppressor can reverse in 2 to 5 msec. While it is functioning, however, information can only travel in one direction; echoes cannot get back to the sender. Figure 2-20(a) shows the state of the echo suppressors while *A* is talking to *B*. Figure 2-20(b) shows the state after *B* has started talking.

The echo suppressors have several properties that are undesirable for data

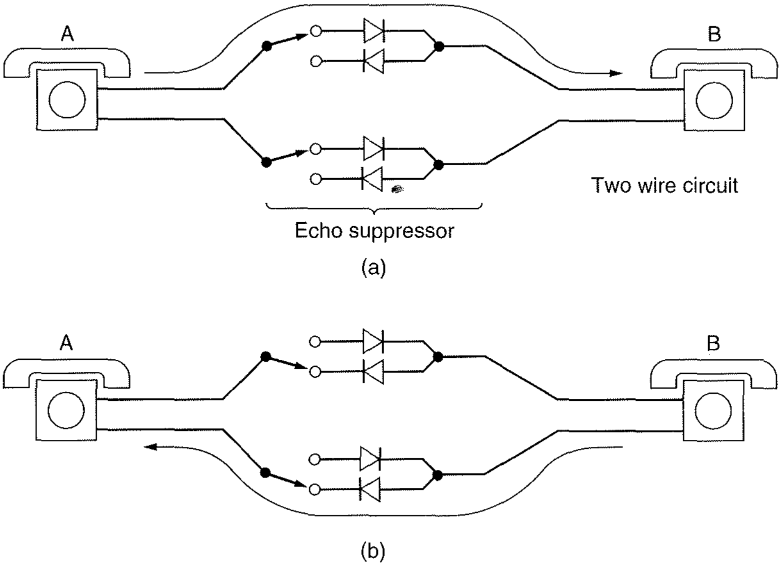


Fig. 2-20. (a) *A* talking to *B*. (b) *B* talking to *A*.

communication. First, it they were not present, it would be possible to transmit in both directions at the same time by using a different frequency band for each direction. This approach is called **full-duplex** transmission. With echo suppressors, full-duplex transmission is impossible. The alternative is **half-duplex** transmission, in which communication can go either way, but only one at a time. A single railroad track is half-duplex. Even if half-duplex transmission is adequate, it is a nuisance because the time required to switch directions can be substantial. Furthermore, the echo suppressors are designed to reverse upon detecting human speech, not digital data.

To alleviate these problems, an escape hatch has been provided on telephone circuits with echo suppressors. When the echo suppressors hear a pure tone at 2100 Hz, they shut down and remain shut down as long as a carrier is present. This arrangement is one of the many examples of **in-band signaling**, so called because the control signals that activate and deactivate internal control functions lie within the band accessible to the user. In general the trend is away from in-band signaling, to prevent users from interfering with the operation of the system itself. In the United States, most of the in-band signaling is gone, but in other countries it still exists.

An alternative to echo suppressors are **echo cancelers.** These are circuits that simulate the echo, estimate how much it is, and subtract it from the signal delivered, without the need for mechanical relays. When echo cancelers are used, full-duplex operation is possible. For this reason, echo cancelers are rapidly replacing echo suppressors in the United States and other large countries.

RS-232-C and RS-449

The interface between the computer or terminal and the modem is an example of a physical layer protocol. It must specify in detail the mechanical, electrical, functional, and procedural interface. We will now look closely at two well-known physical layer standards: RS-232-C and its successor, RS-449.

Let us start with **RS-232-C**, the third revision of the original RS-232 standard. The standard was drawn up by the Electronic Industries Association, a trade organization of electronics manufacturers, and is properly referred to as EIA RS-232-C. The international version is given in CCITT recommendation **V.24**, which is similar but differs slightly on some of the rarely used circuits. In the standards, the terminal or computer is officially called a **DTE** (**Data Terminal Equipment**) and the modem is officially called a **DCE** (**Data Circuit-Terminating Equipment**).

The mechanical specification is for a 25-pin connector 47.04 ± .13 mm wide (screw center to screw center), with all the other dimensions equally well specified. The top row has pins numbered 1 to 13 (left to right); the bottom row has pins numbered 14 to 25 (also left to right).

The electrical specification for RS-232-C is that a voltage more negative than −3 volts is a binary 1 and a voltage more positive than +4 volts is a binary 0. Data rates up to 20 kbps are permitted, as are cables up to 15 meters.

The functional specification tells which circuits are connected to each of the 25 pins, and what they mean. Figure 2-21 shows 9 pins that are nearly always implemented. The remaining ones are frequently omitted. When the terminal or computer is powered up, it asserts (i.e., sets to a logical 1) Data Terminal Ready (pin 20). When the modem is powered up, it asserts Data Set Ready (pin 6). When the modem detects a carrier on the telephone line, it asserts Carrier Detect (pin 8). Request to Send (pin 4) indicates that the terminal wants to send data. Clear to Send (pin 5) means that the modem is prepared to accept data. Data are transmitted on the Transmit circuit (pin 2) and received on the Receive circuit (pin 3).

Other circuits are provided for selecting the data rate, testing the modem, clocking the data, detecting ringing signals, and sending data in the reverse direction on a secondary channel. They are hardly ever used in practice.

The procedural specification is the protocol, that is, the legal sequence of events. The protocol is based on action-reaction pairs. When the terminal asserts Request to Send, for example, the modem replies with Clear to Send, if it is able to accept data. Similar action-reaction pairs exist for other circuits as well.

It commonly occurs that two computers must be connected using RS-232-C. Since neither one is a modem, there is an interface problem. This problem is solved by connecting them with a device called a **null modem**, which connects the transmit line of one machine to the receive line of the other. It also crosses some of the other lines in a similar way. A null modem looks like a short cable.

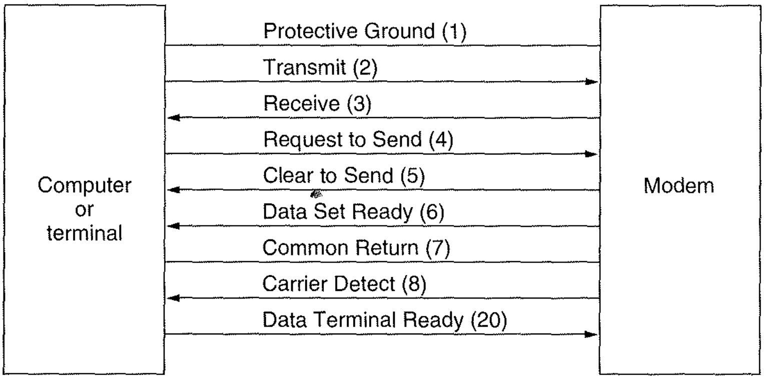


Fig. 2-21. Some of the principal RS-232-C circuits. The pin numbers are given in parentheses.

RS-232-C has been around for years. Gradually, the limitation of the data rate to not more than 20 kbps and the 15-meter maximum cable length have become increasingly annoying. EIA had a long debate about whether to try to have a new standard that was compatible with the old one (but technically not very advanced) or a new and incompatible one that would meet all needs for years to come. They eventually compromised by choosing both.

The new standard, called **RS-449**, is actually three standards in one. The mechanical, functional, and procedural interfaces are given in RS-449, but the electrical interface is given by two different standards. The first of these, **RS-423-A**, is similar to RS-232-C in that all its circuits share a common ground. This technique is called **unbalanced transmission**. The second electrical standard, **RS-422-A**, in contrast, uses **balanced transmission**, in which each of the main circuits requires two wires, with no common ground. As a result, RS-422-A can be used at speeds up to 2 Mbps over 60-meter cables.

The circuits used in RS-449 are shown in Fig. 2-22. Several new circuits not present in RS-232-C have been added. In particular, circuits for testing the modem both locally and remotely were included. Due to the inclusion of a number of two-wire circuits (when RS-422-A is used), more pins are needed in the new standard, so the familiar 25-pin connector was dropped. In its place is a 37-pin connector and a 9-pin connector. The 9-pin connector is required only if the second (reverse) channel is being used.

Fiber in the Local Loop

For advanced future services, such as video on demand, the 3-kHz channel currently used will not do. Discussions about what to do about this tend to focus on two solutions. The straightforward one—running a fiber from the end office

	RS-232-C			CCITT V .24			RS-449		
	Code	Pin	Circuit	Code	Pin	Circuit	Code	Pin	Circuit
	AA	1	Protective ground	101	1	Protective ground	–	1	
	AB	7	Signal ground	102	7	Signal ground	SG	19	Signal ground
							SC	37	Send common
							RC	20	Receive common
	BA	2	Transmitted data	103	2	Transmitted data	SD	4, 22	Send data
	BB	3	Received data	104	3	Received data	RD	6, 24	Receive data
	CA	4	Request to send	105	4	Request to send	RS	7, 25	Request to send
	CB	5	Clear to send	106	5	Ready for sending	CS	9, 27	Clear to send
	CC	6	Data set ready	107	6	Data set ready	DM	11, 29	Data mode
	CD	20	Data terminal ready	108	20	Data terminal ready	TR	12, 30	Terminal ready
	CE	22	Ring indicator	125	22	Calling indicator	IC	15	Incoming call
	CF	8	Line detector	109	8	Line detector	RR	13, 31	Receiver ready
	CG	21	Signal quality	110	21	Signal quality	SQ	33	Signal quality
	CH	23	DTE rate	111	23	DTE rate	SR	16	Signaling rate
	CI	18	DCE rate	112	18	DCE rate	SI	2	Signaling indicators
							IS	28	Terminal in service
				136		New signal	NS	34	New signal
				126	11	Select frequency	SF	16	Select frequency
	DA	24	DTE timing	113	24	DTE timing	TT	17, 25	Terminal timing
	DB	15	DCE timing	114	15	DCE timing	ST	5, 23	Send timing
	DD	17	Receiver timing	115	17	Receiver timing	RT	8, 26	Receive timing
Secondary Channel	SBA	14	Transmitted data	118	14	Transmitted data	SSD	3	Send data
	SBB	16	Received data	119	16	Received data	SRD	4	Receive data
	SCA	19	Request to send	120	19	Line signal	SRS	7	Request to send
	SCB	13	Clear to send	121	13	Channel ready	SCS	8	Clear to send
	SCF	12	Line detector	122	12	Line detector	SRR	2	Receiver ready
							LL	10	Local loopback
							RL	14	Remote loopback
							TM	18	Test mode
							SS	32	Select standby
							SB	36	Standby indicator

Fig. 2-22. Comparison of RS-232-C, V.24, and RS-449.

into everyone's house is called **FTTH** (**Fiber To The Home**). This solution fits in well with the current system but will not be economically feasible for decades. It is simply too expensive.

An alternative solution that is much cheaper is **FTTC** (**Fiber To The Curb**). In this model, the telephone company runs an optical fiber from each end office into each neighborhood (the curb) that it serves (Paff, 1995). The fiber is terminated in a junction box that all the local loops enter. Since the local loops are now much shorter (perhaps 100 meters instead of 3 km), they can be run at higher speeds, probably around 1 Mbps, which is just enough for compressed video. This design is shown in Fig. 2-23(a).

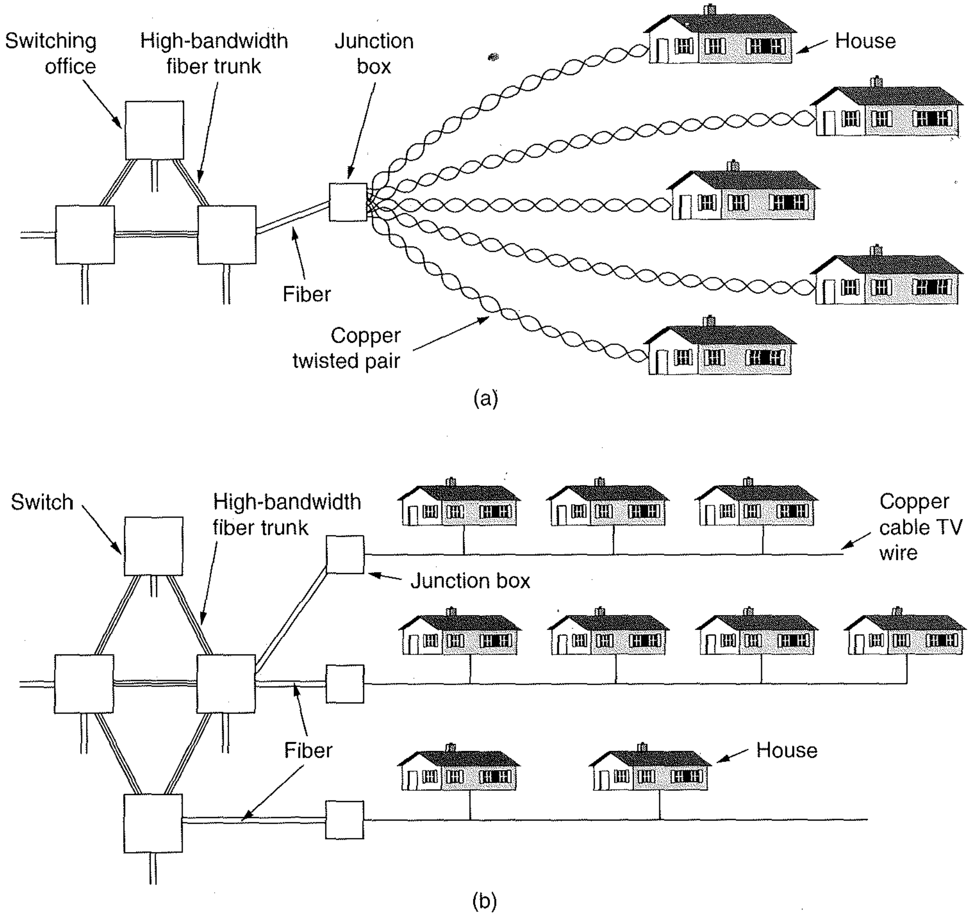


Fig. 2-23. Fiber to the curb. (a) Using the telephone network. (b) Using the cable TV network.

In this manner, multiple videos (or other information channels) can pour down the fiber at high speed and be split over the twisted pairs at the end. By sharing a 1-Gbps fiber over 100 to 1000 customers, the cost per customer can be reduced, and considerably higher bandwidth can be provided than now. Going appreciably above 1 Mbps for long distances with the existing twisted pairs is impossible. Thus in the long term, all the twisted pairs will have to be replaced by fiber. Whether the intermediate solution of FTTC should be used for the time being or

FTTH should be the goal from the beginning is a matter of some debate within the telephone industry.

An alternative design using the existing cable TV infrastructure is shown in Fig. 2-23(b). Here a multidrop cable is used instead of the point-to-point system characteristic of the telephone system. It is likely that both Fig. 2-23(a) and Fig. 2-23(b) will coexist in the future, as telephone companies and cable TV operators become direct competitors for voice, data, and possibly even television service. For more information about this topic, see (Cook and Stern, 1994; Miki, 1994b; and Mochida, 1994).

2.4.4. Trunks and Multiplexing

Economies of scale play an important role in the telephone system. It costs essentially the same amount of money to install and maintain a high-bandwidth trunk as a low-bandwidth trunk between two switching offices (i.e., the costs come from having to dig the trench and not from the copper wire or optical fiber). Consequently, telephone companies have developed elaborate schemes for multiplexing many conversations over a single physical trunk. These multiplexing schemes can be divided into two basic categories: **FDM** (**Frequency Division Multiplexing**), and **TDM** (**Time Division Multiplexing**). In FDM the frequency spectrum is divided among the logical channels, with each user having exclusive possession of some frequency band. In TDM the users take turns (in a round robin), each one periodically getting the entire bandwidth for a little burst of time.

AM radio broadcasting provides illustrations of both kinds of multiplexing. The allocated spectrum is about 1 MHz, roughly 500 to 1500 kHz. Different frequencies are allocated to different logical channels (stations), each operating in a portion of the spectrum, with the interchannel separation great enough to prevent interference. This system is an example of frequency division multiplexing. In addition (in some countries), the individual stations have two logical subchannels: music and advertising. These two alternate in time on the same frequency, first a burst of music, then a burst of advertising, then more music, and so on. This situation is time division multiplexing.

Below we will examine frequency division multiplexing. After that we will see how FDM can be applied to fiber optics (wavelength division multiplexing). Then we will turn to TDM, and end with an advanced TDM system used for fiber optics (SONET).

Frequency Division Multiplexing

Figure 2-24 shows how three voice-grade telephone channels are multiplexed using FDM. Filters limit the usable bandwidth to about 3000 Hz per voice-grade channel. When many channels are multiplexed together, 4000 Hz is allocated to each channel to keep them well separated. First the voice channels are raised in repeaters, which just amplify and regenerate the bits, but do not change or process them in any way.

The line sublayer is concerned with multiplexing multiple tributaries onto a single line and demultiplexing them at the other end. To the line sublayer, the repeaters are transparent. When a multiplexer puts out bits on a fiber, it expects them to arrive at the next multiplexer unchanged, no matter how many repeaters are used in between. The protocol in the line sublayer is thus between two multiplexers and deals with issues such as how many inputs are being multiplexed together and how. In contrast, the path sublayer and protocol deal with end-to-end issues.

2.4.5. Switching

From the point of view of the average telephone engineer, the phone system is divided into two parts: outside plant (the local loops and trunks, since they are outside the switching offices), and inside plant (the switches). We have just looked at outside plant. Now it is time to examine inside plant.

Two different switching techniques are used inside the telephone system: circuit switching and packet switching. We will give a brief introduction to each of them below. Then we will go into circuit switching in detail, because that is how the current telephone system works. Later in the chapter we will go into packet switching in detail in the context of the next generation telephone system, broadband ISDN.

Circuit Switching

When you or your computer places a telephone call, the switching equipment within the telephone system seeks out a physical "copper" (including fiber and radio) path all the way from your telephone to the receiver's telephone. This technique is called **circuit switching** and is shown schematically in Fig. 2-34(a). Each of the six rectangles represents a carrier switching office (end office, toll office, etc.). In this example, each office has three incoming lines and three outgoing lines. When a call passes through a switching office, a physical connection is (conceptually) established between the line on which the call came in and one of the output lines, as shown by the dotted lines.

In the early days of the telephone, the connection was made by having the operator plug a jumper cable into the input and output sockets. In fact, there is a surprising little story associated with the invention of automatic circuit switching equipment. It was invented by a 19th Century undertaker named Almon B. Strowger. Shortly after the telephone was invented, when someone died, one of the survivors would call the town operator and say: "Please connect me to an undertaker." Unfortunately for Mr. Strowger, there were two undertakers in his town, and the other one's wife was the town telephone operator. He quickly saw that either he was going to have to invent automatic telephone switching equipment or he was going to go out of business. He chose the first option. For nearly 100 years, the circuit switching equipment used worldwide was known as Strowger gear. (History does not record whether the now-unemployed switchboard operator got a job as an information operator, answering questions such as: What is the phone number of an undertaker?

The model shown in Fig. 2-34(a) is highly simplified of course, because parts of the "copper" path between the two telephones may, in fact, be microwave links onto which thousands of calls are multiplexed. Nevertheless, the basic idea is valid: once a call has been set up, a dedicated path between both ends exists and will continue to exist until the call is finished.

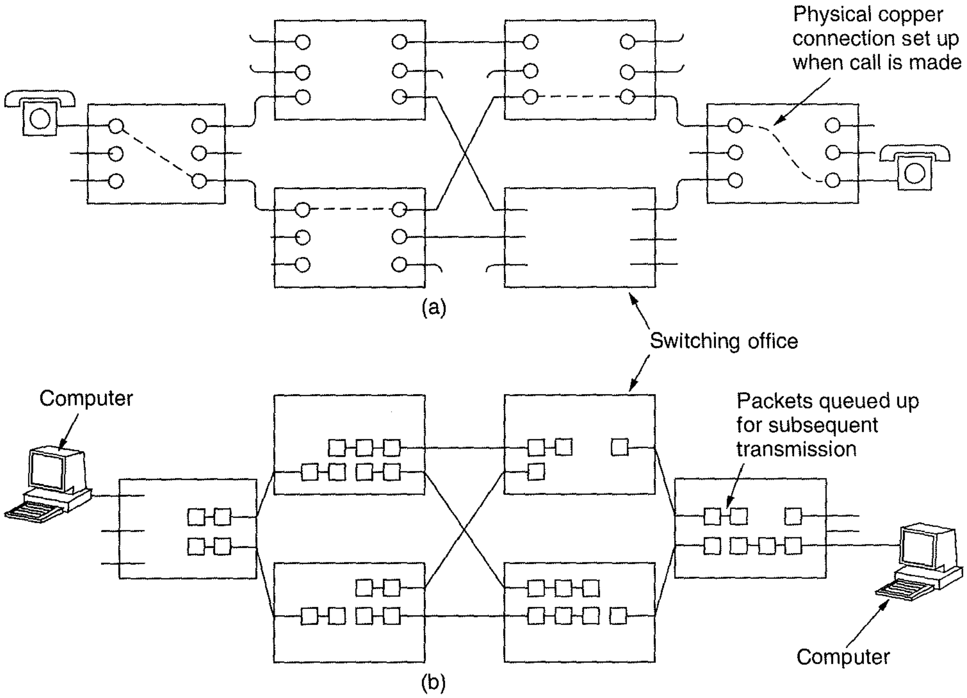


Fig. 2-34. (a) Circuit switching. (b) Packet switching.

An important property of circuit switching is the need to set up an end-to-end path *before* any data can be sent. The elapsed time between the end of dialing and the start of ringing can easily be 10 sec, more on long-distance or international calls. During this time interval, the telephone system is hunting for a copper path, as shown in Fig. 2-35(a). Note that before data transmission can even begin, the call request signal must propagate all the way to the destination, and be acknowledged. For many computer applications (e.g., point-of-sale credit verification), long setup times are undesirable.

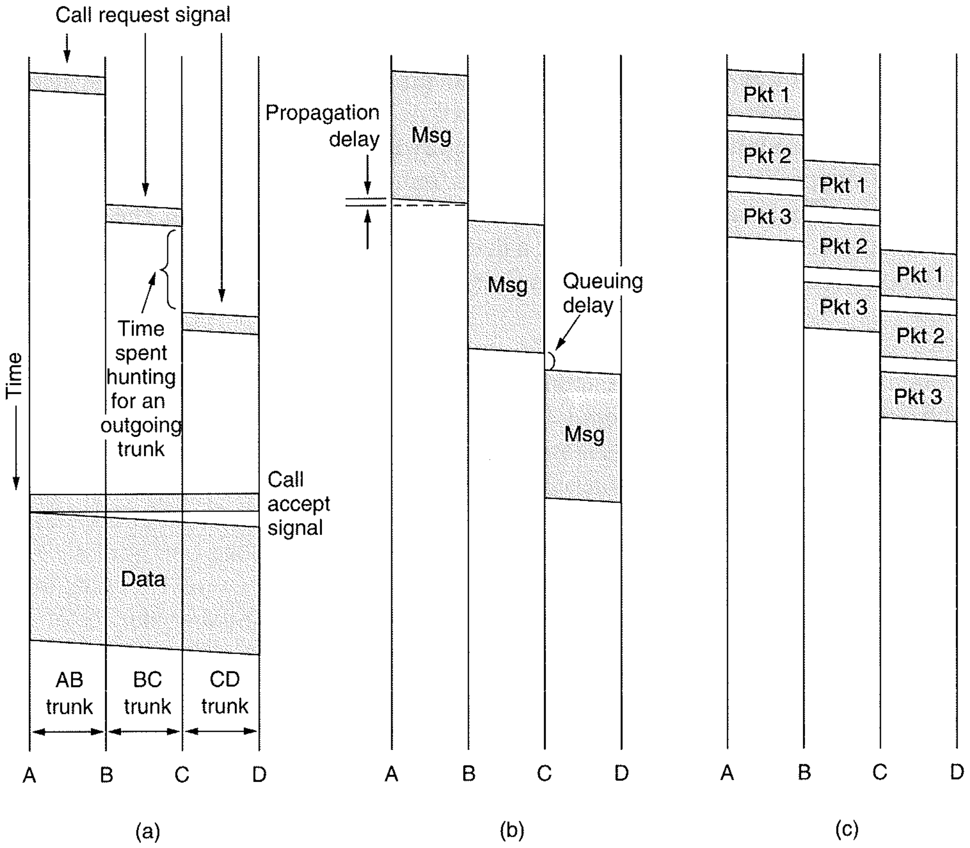


Fig. 2-35. Timing of events in (a) circuit switching, (b) message switching, (c) packet switching.

As a consequence of the copper path between the calling parties, once the setup has been completed, the only delay for data is the propagation time for the electromagnetic signal, about 5 msec per 1000 km. Also as a consequence of the established path, there is no danger of congestion—that is, once the call has been put through, you never get busy signals, although you might get one before the connection has been established due to lack of switching or trunk capacity.

An alternative switching strategy is **message switching**, shown in Fig. 2-35(b). When this form of switching is used, no physical copper path is established in advance between sender and receiver. Instead, when the sender has a block of data to be sent, it is stored in the first switching office (i.e., router) and then forwarded later, one hop at a time. Each block is received in its entirety, inspected for errors, and then retransmitted. A network using this technique is called a **store-and-forward** network, as mentioned in Chap. 1.

The first electromechanical telecommunication systems used message switching, namely for telegrams. The message was punched on paper tape off-line at the sending office, and then read in and transmitted over a communication line to the next office along the way, where it was punched out on paper tape. An operator there tore the tape off and read it in on one of the many tape readers, one per outgoing trunk. Such a switching office was called a **torn tape office**.

With message switching, there is no limit on block size, which means that routers (in a modern system) must have disks to buffer long blocks. It also means that a single block may tie up a router-router line for minutes, rendering message switching useless for interactive traffic. To get around these problems, **packet switching** was invented. Packet-switching networks place a tight upper limit on block size, allowing packets to be buffered in router main memory instead of on disk. By making sure that no user can monopolize any transmission line very long (milliseconds), packet-switching networks are well suited to handling interactive traffic. A further advantage of packet switching over message switching is shown in Fig. 2-35(b) and (c): the first packet of a multipacket message can be forwarded before the second one has fully arrived, reducing delay and improving throughput. For these reasons, computer networks are usually packet switched, occasionally circuit switched, but never message switched.

Circuit switching and packet switching differ in many respects. The key difference is that circuit switching statically reserves the required bandwidth in advance, whereas packet switching acquires and releases it as it is needed. With circuit switching, any unused bandwidth on an allocated circuit is just wasted. With packet switching it may be utilized by other packets from unrelated sources going to unrelated destinations, because circuits are never dedicated. However, just because no circuits are dedicated, a sudden surge of input traffic may overwhelm a router, exceeding its storage capacity and causing it to lose packets.

In contrast, with circuit switching, when packet switching is used, it is straightforward for the routers to provide speed and code conversion. Also, they can provide error correction to some extent. In some packet-switched networks, however, packets may be delivered in the wrong order to the destination. Reordering of packets can never happen with circuit switching.

Another difference is that circuit switching is completely transparent. The sender and receiver can use any bit rate, format, or framing method they want to. The carrier does not know or care. With packet switching, the carrier determines the basic parameters. A rough analogy is a road versus a railroad. In the former, the user determines the size, speed, and nature of the vehicle; in the latter, the carrier does. It is this transparency that allows voice, data, and fax to coexist within the phone system.

A final difference between circuit and packet switching is the charging algorithm. Packet carriers usually base their charge on both the number of bytes (or packets) carried and the connect time. Furthermore, transmission distance usually does not matter, except perhaps internationally. With circuit switching, the charge is based on the distance and time only, not the traffic. The differences are summarized in Fig. 2-36.

Item	Circuit-switched	Packet-switched
Dedicated "copper" path	Yes	No
Bandwidth available	Fixed	Dynamic
Potentially wasted bandwidth	Yes	No
Store-and-forward transmission	No	Yes
Each packet follows the same route	Yes	No
Call setup	Required	Not needed
When can congestion occur	At setup time	On every packet
Charging	Per minute	Per packet

Fig. 2-36. A comparison of circuit-switched and packet-switched networks.

Both circuit switching and packet switching are so important, we will come back to them shortly and describe the various technologies used in detail.

The Switch Hierarchy

It is worth saying a few words about how the routing between switches is done within the current circuit-switched telephone system. We will describe the AT&T system here, but other companies and countries use the same general principles. The telephone system has five classes of switching offices, as illustrated in Fig. 2-37. There are 10 regional switching offices, and these are fully interconnected by 45 high-bandwidth fiber optic trunks. Below the regional offices are 67 sectional offices, 230 primary offices, 1300 toll offices, and 19,000 end offices. The lower four levels were originally connected as a tree.

Calls are generally connected at the lowest possible level. Thus if a subscriber connected to end office 1 calls another subscriber connected to end office 1, the call will be completed in that office. However, a call from a customer attached to end office 1 in Fig. 2-37 to a customer attached to end office 2 will have to go toll office 1. However, a call from end office 1 to end office 4 will have to go up to primary office 1, and so on. With a pure tree, there is only one minimal route, and that would normally be taken.

During years of operation, the telephone companies noticed that some routes were busier than others. For example, there were many calls from New York to Los Angeles. Rather than go all the way up the hierarchy, they simply installed **direct trunks** for the busy routes. A few of these are shown in Fig. 2-37 as

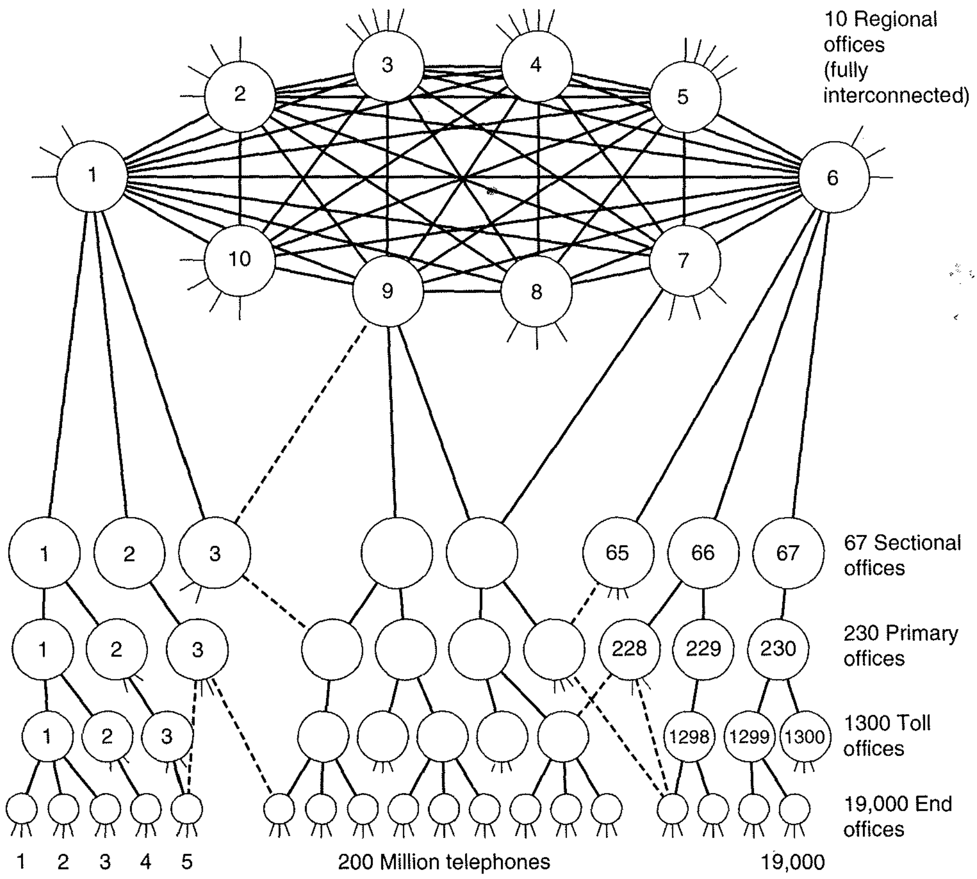


Fig. 2-37. The AT&T telephone hierarchy. The dashed lines are direct trunks.

dashed lines. As a consequence, many calls can now be routed along many paths. The actual route chosen is generally the most direct one, but if the necessary trunks along it are full, an alternative is chosen. This complex routing is now possible because a switching machine, like the AT&T 5 ESS, is in fact just a general purpose computer with a large amount of very specialized I/O equipment.

Crossbar Switches

Let us now turn from how calls are routed among switches to how individual switches actually work inside. Several kinds of switches are (or were) common within the telephone system. The simplest kind is the **crossbar switch** (also called a **crosspoint switch**), shown in Fig. 2-38. In a switch with n input lines and n output lines (i.e., n full duplex lines), the crossbar switch has n^2

to line 0, and so on. In essence, the switch has moved a byte from input line 4 to output line 0, another byte from input line 7 to output line 1, and so on. Viewed from the outside, the whole arrangement is a circuit switch, even though there are no physical connections.

The time slot interchanger works as follows: When an input frame is ready to be processed, each slot (i.e., each byte in the input frame) is written into a RAM buffer inside the interchanger. The slots are written in order, so buffer word i contains slot i.

After all the slots of the input frame have been stored in the buffer, the output frame is constructed by reading out the words again, but in a different order. A counter goes from 0 to $n - 1$. At step j, the contents of word j of a mapping table is read out and used to address the RAM table. Thus if word 0 of the mapping table contains a 4, word 4 of the RAM buffer will be read out first, and the first slot of the output frame will be slot 4 of the input frame. Thus the contents of the mapping table determine which permutation of the input frame will be generated as the output frame, and thus which input line is connected to which output line.

Time division switches use tables that are linear in the number of lines, rather than quadratic, but they have another limitation. It is necessary to store n slots in the buffer RAM and then read them out again within one frame period of 125 μsec. If each of these memory accesses takes T microsec, the time needed to process a frame is $2nT$ microsec, so we have $2nT = 125$ or $n = 125/2T$. For a memory with 100-nsec cycle time, we can support at most 625 lines. We can also turn this relation around and use it to determine the required memory cycle to support a given number of lines. As with a crossbar switch, it is possible to devise multistage switches that split the work up into several parts and then combine the results in order to handle larger numbers of lines.

2.5. NARROWBAND ISDN

For more than a century, the primary international telecommunication infrastructure has been the public circuit-switched telephone system. This system was designed for analog voice transmission and is inadequate for modern communication needs. Anticipating considerable user demand for an end-to-end digital service (i.e., not like Fig. 2-17 which is part digital and part analog), the world's telephone companies and PTTs got together in 1984 under the auspices of CCITT and agreed to build a new, fully digital, circuit-switched telephone system by the early part of the 21st Century. This new system, called **ISDN** (**Integrated Services Digital Network**), has as its primary goal the integration of voice and nonvoice services. It is already available in many locations and its use is growing slowly. In the following sections we will describe what it does and how it works. For further information, see (Dagdeviren et al., 1994; and Kessler, 1993).

2.5.1. ISDN Services

The key ISDN service will continue to be voice, although many enhanced features will be added. For example, many corporate managers have an intercom button on their telephone that rings their secretaries instantly (no call setup time). One ISDN feature is telephones with multiple buttons for instant call setup to arbitrary telephones anywhere in the world. Another feature is telephones that display the caller's telephone number, name, and address on a display while ringing. A more sophisticated version of this feature allows the telephone to be connected to a computer, so that the caller's database record is displayed on the screen as the call comes in. For example, a stockbroker could arrange that when she answers the telephone, the caller's portfolio is already on the screen along with the current prices of all the caller's stocks. Other advanced voice services include call forwarding and conference calls worldwide.

Advanced nonvoice services are remote electricity meter reading, and on-line medical, burglar, and smoke alarms that automatically call the hospital, police, or fire department, respectively, and give their address to speed up response.

2.5.2. ISDN System Architecture

It is now time to look at the ISDN architecture in detail, particularly the customer's equipment and the interface between the customer and the telephone company or PTT. The key idea behind ISDN is that of the **digital bit pipe**, a conceptual pipe between the customer and the carrier through which bits flow. Whether the bits originated from a digital telephone, a digital terminal, a digital facsimile machine, or some other device is irrelevant. All that matters is that bits can flow through the pipe in both directions.

The digital bit pipe can, and normally does, support multiple independent channels by time division multiplexing of the bit stream. The exact format of the bit stream and its multiplexing is a carefully defined part of the interface specification for the digital bit pipe. Two principal standards for the bit pipe have been developed, a low bandwidth standard for home use and a higher bandwidth standard for business use that supports multiple channels that are identical to the home use channel. Furthermore, businesses may have multiple bit pipes if they need additional capacity beyond what the standard business pipe can provide.

In Fig. 2-41(a) we see the normal configuration for a home or small business. The carrier places a network terminating device, **NT1**, on the customer's premises and connects it to the ISDN exchange in the carrier's office, several kilometers away, using the twisted pair that was previously used to connect to the telephone. The NT1 box has a connector on it into which a passive bus cable can be inserted. Up to eight ISDN telephones, terminals, alarms, and other devices can be connected to the cable, similar to the way devices are connected to a LAN. From the customer's point of view, the network boundary is the connector on NT1.

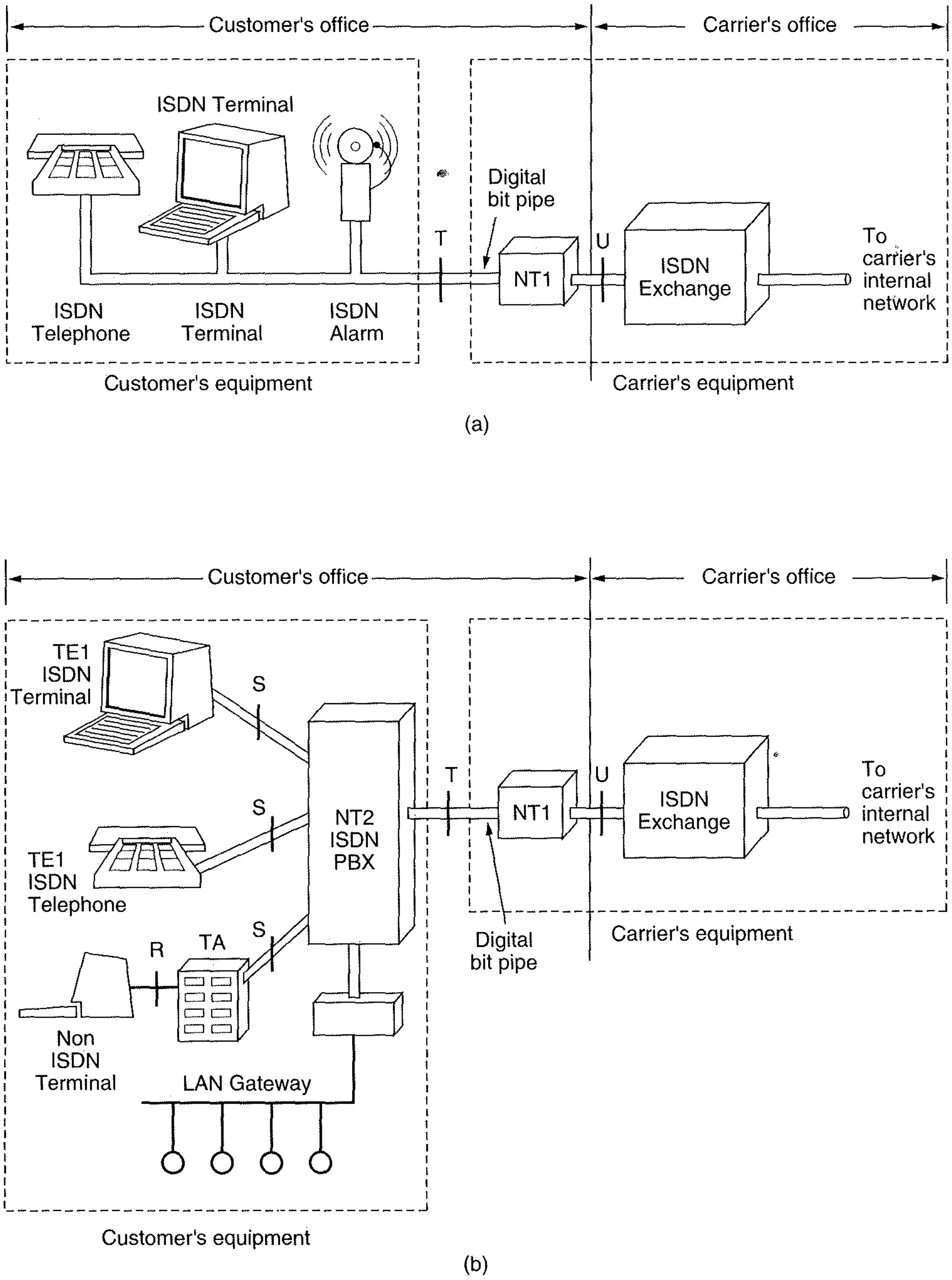


Fig. 2-41. (a) Example ISDN system for home use. (b) Example ISDN system with a PBX for use in large businesses.

For large businesses, the model of Fig. 2-41(a) is inadequate because it is common to have more telephone conversations going on simultaneously than the bus can handle. Therefore, the model of Fig. 2-41(b) is used. In this model we find a device, **NT2**, called a **PBX** (**Private Branch eXchange**), connected to NT1 and providing the real interface for telephones, terminals and other equipment. An ISDN PBX is not very different conceptually from an ISDN switch, although it is usually smaller and cannot handle as many conversations at the same time.

CCITT defined four **reference points**, called **R**, **S**, **T**, and **U**, between the various devices. These are marked in Fig. 2-41. The U reference point is the connection between the ISDN exchange in the carrier's office and NT1. At present it is a two-wire copper twisted pair, but at some time in the future it may be replaced by fiber optics. The T reference point is what the connector on NT1 provides to the customer. The S reference point is the interface between the ISDN PBX and the ISDN terminals. The R reference point is the connection between the terminal adapter and non-ISDN terminals. Many different kinds of interfaces will be used at R.

2.5.3. The ISDN Interface

The ISDN bit pipe supports multiple channels interleaved by time division multiplexing. Several channel types have been standardized:

A - 4-kHz analog telephone channel
B - 64-kbps digital PCM channel for voice or data
C - 8- or-16 kbps digital channel
D - 16-kbps digital channel for out-of-band signaling
E - 64-kbps digital channel for internal ISDN signaling
H - 384-, 1536-, or 1920-kbps digital channel

It was not CCITT's intention to allow an arbitrary combination of channels on the digital bit pipe. Three combinations have been standardized so far:

1. **Basic rate**: 2B + 1D
2. **Primary rate**: 23B + 1D (U.S. and Japan) or 30B + 1D (Europe)
3. **Hybrid**: 1A + 1C

The basic rate and primary rate channels are illustrated in Fig. 2-42.

The basic rate should be viewed as a replacement for **POTS** (**Plain Old Telephone Service**) for home or small business use. Each of the 64-kbps B channels can handle a single PCM voice channel with 8-bit samples made 8000 times a second (note that 64 kbps means 64,000 here, not 65,536). Signaling is on a separate 16-kbps D channel, so the full 64 kbps are available to the user (as in the CCITT 2.048-Mbps system and unlike the U.S. and Japanese T1 system).

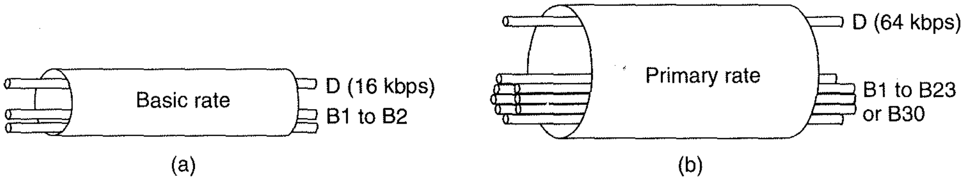


Fig. 2-42. (a) Basic rate digital pipe. (b) Primary rate digital pipe.

Because ISDN is so focused on 64-kbps channels, we refer to it as **N-ISDN** (**Narrowband ISDN**), to contrast it with broadband ISDN (ATM) to be discussed later.

The primary rate interface is intended for use at the T reference point for businesses with a PBX. It has 23 B channels and 1 D channel (at 64 kbps) in the United States and Japan and 30 B channels and 1 D channel (at 64 kbps) in Europe. The 23B + 1D choice was made to allow an ISDN frame fit nicely on AT&T's T1 system. The 30B + 1D choice was made to allow an ISDN frame fit nicely in CCITT's 2.048 Mbps system. The 32nd time slot in the CCITT system is used for framing and general network maintenance. Note that the amount of D channel per B channel in the primary rate is much less than in the basic rate, as it is not expected that there will be much telemetry or low bandwidth packet data there.

2.5.4. Perspective on N-ISDN

N-ISDN was a massive attempt to replace the analog telephone system with a digital one suitable for both voice and nonvoice traffic. Achieving worldwide agreement on the interface standard for the basic rate was supposed to lead to a large user demand for ISDN equipment, thus leading to mass production, economies of scale, and inexpensive VLSI ISDN chips. Unfortunately, the standardization process took years and the technology in this area moved very rapidly, so that once the standard was finally agreed upon, it was obsolete.

For home use, the largest demand for new services will undoubtedly be for video on demand. Unfortunately, the ISDN basic rate lacks the necessary bandwidth by two orders of magnitude. For business use, the situation is even bleaker. Currently available LANs offer at least 10 Mbps and are now being replaced by 100-Mbps LANs. Offering 64-kbps service to businesses in the 1980s was a serious proposition. In the 1990s, it is a joke.

Oddly enough, ISDN may yet be saved, but by a totally unexpected application: Internet access. Various companies now sell ISDN adaptors that combine the 2B + D channels into a single 144-kbps digital channel. Many Internet service providers also support these adaptors. The result is that people can access the

Internet over a 144-kbps fully digital link, instead of a 28.8-kbps analog modem link. For many Internet users, gaining a factor of five for downloading World Wide Web pages full of graphics is a service worth having. While B-ISDN at 155 Mbps is even better, N-ISDN at 144 kbps is here now for an affordable price, and that may be its main niche for the next few years.

2.6. BROADBAND ISDN AND ATM

When CCITT finally figured out that narrowband ISDN was not going to set the world on fire, it tried to think of a new service that might. The result was **broadband ISDN (B-ISDN)**, basically a digital virtual circuit for moving fixed-size packets (cells) from source to destination at 155 Mbps (really 156 Mbps, as mentioned earlier). Since this data rate is even enough for (uncompressed) HDTV, it is likely to satisfy even the biggest bandwidth hogs for at least a few years.

Whereas narrowband ISDN was a timid first step into the digital age, broadband ISDN is a bold leap into the unknown. The benefits are enormous, such as a bandwidth increase over narrowband ISDN by a factor of 2500, but the challenges are equally huge (Armbruster, 1995).

To start with, broadband ISDN is based on ATM technology, and as we discussed briefly in Chap. 1, ATM is fundamentally a packet-switching technology, not a circuit-switching technology (although it can emulate circuit switching fairly well). In contrast, both the existing PSTN and narrowband ISDN are circuit-switching technologies. An enormous amount of engineering experience in circuit switching will be rendered obsolete by this change. Going from circuit switching to packet switching is truly a paradigm shift.

As if that were not enough, broadband ISDN cannot be sent over existing twisted pair wiring for any substantial distance. This means that introducing it will require ripping out most of the local loops and putting in either category 5 twisted pair or fiber (Stephens and Banwell, 1995). Furthermore, space division and time division switches cannot be used for packet switching. They will all have to be replaced by new switches based on different principles and running at much higher speeds. The only things that can be salvaged are the wide area fiber trunks.

In short, throwing out 100 years' accumulated knowledge plus an investment in both inside plant and outside plant worth many hundreds of billions of dollars is not exactly a small step to be taken lightly. Nevertheless, it is clear to the telephone companies that if they do not do it, the cable television companies, thinking about video on demand, probably will. While it is likely that both the existing PSTN and narrowband ISDN will be around for a decade or perhaps even longer, the long-term future probably lies with ATM, so we will study it in great detail in this book, starting with the physical layer in this chapter.

3.6.2. The Data Link Layer in the Internet

The Internet consists of individual machines (hosts and routers), and the communication infrastructure that connects them. Within a single building, LANs are widely used for interconnection, but most of the wide area infrastructure is built up from point-to-point leased lines. In Chap. 4, we will look at LANs; here we will examine the data link protocols used on point-to-point lines in the Internet.

In practice, point-to-point communication is primarily used in two situations. First, thousands of organizations have one or more LANs, each with some number of hosts (personal computers, user workstations, servers, and so on) along with a router (or a bridge, which is functionally similar). Often, the routers are interconnected by a backbone LAN. Typically, all connections to the outside world go through one or two routers that have point-to-point leased lines to distant routers. It is these routers and their leased lines that make up the communication subnets on which the Internet is built.

The second situation where point-to-point lines play a major role in the Internet is the millions of individuals who have home connections to the Internet using modems and dial-up telephone lines. Usually, what happens is that the user's home PC calls up an **Internet provider**, which includes commercial companies like America Online, CompuServe, and the Microsoft Network, but also many universities and companies that provide home Internet connectivity to their students and employees. Sometimes the home PC just functions as a character-oriented terminal logged into the Internet service provider's timesharing system. In this mode, the user can type commands and run programs, but the graphical Internet services, such as the World Wide Web, are not available. This way of working is called having a **shell account**.

Alternatively, the home PC can call an Internet service provider's router and then act like a full-blown Internet host. This method of operation is no different than having a leased line between the PC and the router, except that the connection is terminated when the user ends the session. With this approach, all Internet services, including the graphical ones, become available. A home PC calling an Internet service provider is illustrated in Fig. 3-26.

For both the router-router leased line connection and the dial-up host-router connection, some point-to-point data link protocol is required on the line for framing, error control, and the other data link layer functions we have studied in this chapter. Two such protocols are widely used in the Internet, SLIP and PPP. We will now examine each of these in turn.

SLIP—Serial Line IP

SLIP is the older of the two protocols. It was devised by Rick Adams in 1984 to connect Sun workstations to the Internet over a dial-up line using a modem. The protocol, which is described in RFC 1055, is very simple. The workstation

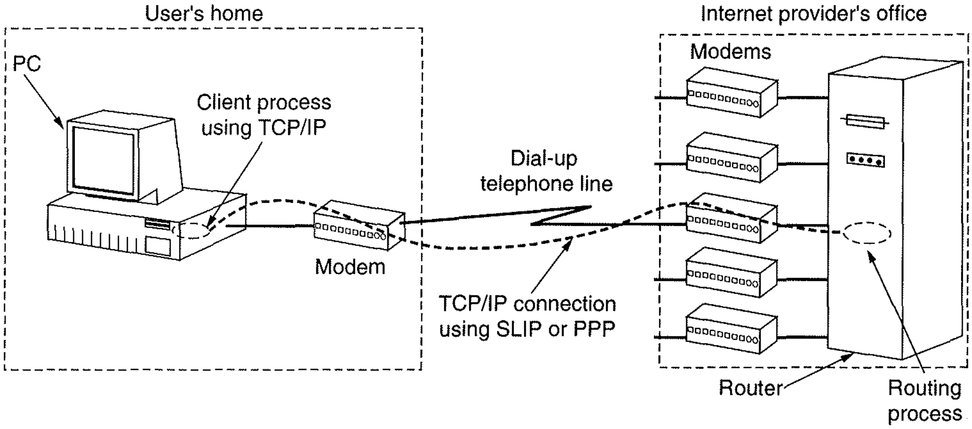


Fig. 3-26. A home personal computer acting as an Internet host.

just sends raw IP packets over the line, with a special flag byte (0xC0) at the end for framing. If the flag byte occurs inside the IP packet, a form of character stuffing is used, and the two byte sequence (0xDB, 0xDC) is sent in its place. If 0xDB occurs inside the IP packet, it, too, is stuffed. Some SLIP implementations attach a flag byte to both the front and back of each IP packet sent.

More recent versions of SLIP do some TCP and IP header compression. What they do is take advantage of the fact that consecutive packets often have many header fields in common. These are compressed by omitting those fields that are the same as the corresponding fields in the previous IP packet. Furthermore, the fields that do differ are not sent in their entirety, but as increments to the previous value. These optimizations are described in RFC 1144.

Although it is still widely used, SLIP has some serious problems. First, it does not do any error detection or correction, so it is up to higher layers to detect and recover from lost, damaged, or merged frames.

Second, SLIP supports only IP. With the growth of the Internet to encompass networks that do not use IP as their native language (e.g., Novell LANs), this restriction is becoming increasingly serious.

Third, each side must know the other's IP address in advance; neither address can be dynamically assigned during setup. Given the current shortage of IP addresses, this limitation is a major issue as it is impossible to give each home Internet user a unique IP address.

Fourth, SLIP does not provide any form of authentication, so neither party knows whom it is really talking to. With leased lines, this is not an issue, but with dial-up lines it is.

Fifth, SLIP is not an approved Internet Standard, so many different (and incompatible) versions exist. This situation does not make interworking easier.

the actions following the wakeup also count. For example, if a CALL REQUEST packet comes in and a process was asleep waiting for it, the transmission of the CALL ACCEPT packet following the wakeup counts as part of the action for CALL REQUEST. After each action is performed, the connection may move to a new state, as shown in Fig. 6-21.

The advantage of representing the protocol as a matrix is threefold. First, in this form it is much easier for the programmer to systematically check each combination of state and event to see if an action is required. In production implementations, some of the combinations would be used for error handling. In Fig. 6-21 no distinction is made between impossible situations and illegal ones. For example, if a connection is in *waiting* state, the DISCONNECT event is impossible because the user is blocked and cannot execute any primitives at all. On the other hand, in *sending* state, data packets are not expected because no credit has been issued. The arrival of a data packet is a protocol error.

The second advantage of the matrix representation of the protocol is in implementing it. One could envision a two-dimensional array in which element $a[i][j]$ was a pointer or index to the procedure that handled the occurrence of event i when in state j. One possible implementation is to write the transport entity as a short loop, waiting for an event at the top of the loop. When an event happens, the relevant connection is located and its state is extracted. With the event and state now known, the transport entity just indexes into the array a and calls the proper procedure. This approach gives a much more regular and systematic design than our transport entity.

The third advantage of the finite state machine approach is for protocol description. In some standards documents, the protocols are given as finite state machines of the type of Fig. 6-21. Going from this kind of description to a working transport entity is much easier if the transport entity is also driven by a finite state machine based on the one in the standard.

The primary disadvantage of the finite state machine approach is that it may be more difficult to understand than the straight programming example we used initially. However, this problem may be partially solved by drawing the finite state machine as a graph, as is done in Fig. 6-22.

6.4. THE INTERNET TRANSPORT PROTOCOLS (TCP AND UDP)

The Internet has two main protocols in the transport layer, a connection-oriented protocol and a connectionless one. In the following sections we will study both of them. The connection-oriented protocol is TCP. The connectionless protocol is UDP. Because UDP is basically just IP with a short header added, we will focus on TCP.

TCP (**Transmission Control Protocol**) was specifically designed to provide a reliable end-to-end byte stream over an unreliable internetwork. An

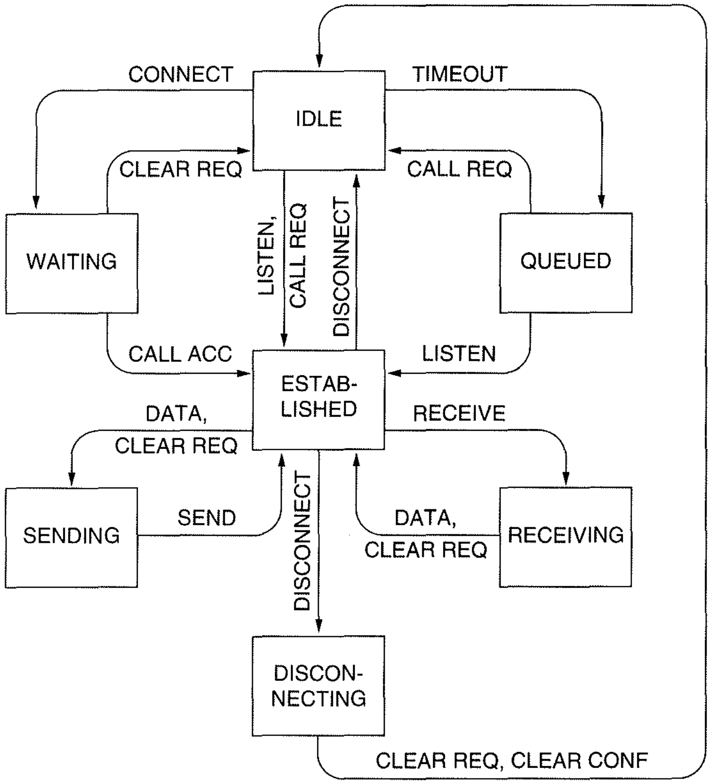


Fig. 6-22. The example protocol in graphical form. Transitions that leave the connection state unchanged have been omitted for simplicity.

internetwork differs from a single network because different parts may have wildly different topologies, bandwidths, delays, packet sizes, and other parameters. TCP was designed to dynamically adapt to properties of the internetwork and to be robust in the face of many kinds of failures.

TCP was formally defined in RFC 793. As time went on, various errors and inconsistencies were detected, and the requirements were changed in some areas. These clarifications and some bug fixes are detailed in RFC 1122. Extensions are given in RFC 1323.

Each machine supporting TCP has a TCP transport entity, either a user process or part of the kernel that manages TCP streams and interfaces to the IP layer. A TCP entity accepts user data streams from local processes, breaks them up into pieces not exceeding 64K bytes (in practice, usually about 1500 bytes), and sends each piece as a separate IP datagram. When IP datagrams containing TCP data arrive at a machine, they are given to the TCP entity, which reconstructs the original byte streams. For simplicity, we will sometimes use just "TCP" to mean the

TCP transport entity (a piece of software) or the TCP protocol (a set of rules). From the context it will be clear which is meant. For example, in "The user gives TCP the data," the TCP transport entity is clearly intended.

The IP layer gives no guarantee that datagrams will be delivered properly, so it is up to TCP to time out and retransmit them as need be. Datagrams that do arrive may well do so in the wrong order; it is also up to TCP to reassemble them into messages in the proper sequence. In short, TCP must furnish the reliability that most users want and that IP does not provide.

6.4.1. The TCP Service Model

TCP service is obtained by having both the sender and receiver create end points, called sockets, as discussed in Sec. 6.1.3. Each socket has a socket number (address) consisting of the IP address of the host and a 16-bit number local to that host, called a **port**. A port is the TCP name for a TSAP. To obtain TCP service, a connection must be explicitly established between a socket on the sending machine and a socket on the receiving machine. The socket calls are listed in Fig. 6-6.

A socket may be used for multiple connections at the same time. In other words, two or more connections may terminate at the same socket. Connections are identified by the socket identifiers at both ends, that is, (*socket1*, *socket2*). No virtual circuit numbers or other identifiers are used.

Port numbers below 256 are called **well-known ports** and are reserved for standard services. For example, any process wishing to establish a connection to a host to transfer a file using FTP can connect to the destination host's port 21 to contact its FTP daemon. Similarly, to establish a remote login session using TELNET, port 23 is used. The list of well-known ports is given in RFC 1700.

All TCP connections are full-duplex and point-to-point. Full duplex means that traffic can go in both directions at the same time. Point-to-point means that each connection has exactly two end points. TCP does not support multicasting or broadcasting.

A TCP connection is a byte stream, not a message stream. Message boundaries are not preserved end to end. For example, if the sending process does four 512-byte writes to a TCP stream, these data may be delivered to the receiving process as four 512-byte chunks, two 1024-byte chunks, one 2048-byte chunk (see Fig. 6-23), or some other way. There is no way for the receiver to detect the unit(s) in which the data were written.

Files in UNIX have this property too. The reader of a file cannot tell whether the file was written a block at a time, a byte at a time, or all in one blow. As with a UNIX file, the TCP software has no idea of what the bytes mean and no interest in finding out. A byte is just a byte.

When an application passes data to TCP, TCP may send it immediately or buffer it (in order to collect a larger amount to send at once), at its discretion.

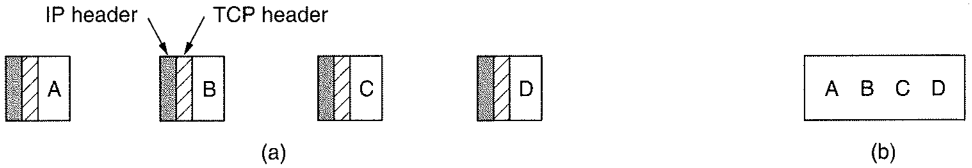


Fig. 6-23. (a) Four 512-byte segments sent as separate IP datagrams. (b) The 2048 bytes of data delivered to the application in a single READ call.

However, sometimes, the application really wants the data to be sent immediately. For example, suppose a user is logged into a remote machine. After a command line has been finished and the carriage return typed, it is essential that the line be shipped off to the remote machine immediately and not buffered until the next line comes in. To force data out, applications can use the PUSH flag, which tells TCP not to delay the transmission.

Some early applications used the PUSH flag as a kind of marker to delineate messages boundaries. While this trick sometimes works, it sometimes fails since not all implementations of TCP pass the PUSH flag to the application on the receiving side. Furthermore, if additional PUSHes come in before the first one has been transmitted (e.g., because the output line is busy), TCP is free to collect all the PUSHed data into a single IP datagram, with no separation between the various pieces.

One last feature of the TCP service that is worth mentioning here is **urgent data**. When an interactive user hits the DEL or CTRL-C key to break off a remote computation that has already begun, the sending application puts some control information in the data stream and gives it to TCP along with the URGENT flag. This event causes TCP to stop accumulating data and transmit everything it has for that connection immediately.

When the urgent data are received at the destination, the receiving application is interrupted (e.g., given a signal in UNIX terms), so it can stop whatever it was doing and read the data stream to find the urgent data. The end of the urgent data is marked, so the application knows when it is over. The start of the urgent data is not marked. It is up to the application to figure that out. This scheme basically provides a crude signaling mechanism and leaves everything else up to the application.

6.4.2. The TCP Protocol

In this section we will give a general overview of the TCP protocol. In the next one we will go over the protocol header, field by field. Every byte on a TCP connection has its own 32-bit sequence number. For a host blasting away at full speed on a 10-Mbps LAN, theoretically the sequence numbers could wrap around in an hour, but in practice it takes much longer. The sequence numbers are used both for acknowledgements and for the window mechanism, which use separate 32-bit header fields.

The sending and receiving TCP entities exchange data in the form of segments. A **segment** consists of a fixed 20-byte header (plus an optional part) followed by zero or more data bytes. The TCP software decides how big segments should be. It can accumulate data from several writes into one segment or split data from one write over multiple segments. Two limits restrict the segment size. First, each segment, including the TCP header, must fit in the 65,535 byte IP payload. Second, each network has a **maximum transfer unit** or **MTU**, and each segment must fit in the MTU. In practice, the MTU is generally a few thousand bytes and thus defines the upper bound on segment size. If a segment passes through a sequence of networks without being fragmented and then hits one whose MTU is smaller than the segment, the router at the boundary fragments the segment into two or more smaller segments.

A segment that is too large for a network that it must transit can be broken up into multiple segments by a router. Each new segment gets its own IP header, so fragmentation by routers increases the total overhead (because each additional segment adds 20 bytes of extra header information in the form of an IP header).

The basic protocol used by TCP entities is the sliding window protocol. When a sender transmits a segment, it also starts a timer. When the segment arrives at the destination, the receiving TCP entity sends back a segment (with data if any exists, otherwise without data) bearing an acknowledgement number equal to the next sequence number it expects to receive. If the sender's timer goes off before the acknowledgement is received, the sender transmits the segment again.

Although this protocol sounds simple, there are many ins and outs that we will cover below. For example, since segments can be fragmented, it is possible that part of a transmitted segment arrives and is acknowledged by the receiving TCP entity, but the rest is lost. Segments can also arrive out of order, so bytes 3072–4095 can arrive but cannot be acknowledged because bytes 2048–3071 have not turned up yet. Segments can also be delayed so long in transit that the sender times out and retransmits them. If a retransmitted segment takes a different route than the original, and is fragmented differently, bits and pieces of both the original and the duplicate can arrive sporadically, requiring a careful administration to achieve a reliable byte stream. Finally, with so many networks making up the Internet, it is possible that a segment may occasionally hit a congested (or broken) network along its path.

TCP must be prepared to deal with these problems and solve them in an efficient way. A considerable amount of effort has gone into optimizing the performance of TCP streams, even in the face of network problems. A number of the algorithms used by many TCP implementations will be discussed below.

6.4.3. The TCP Segment Header

Figure 6-24 shows the layout of a TCP segment. Every segment begins with a fixed-format 20-byte header. The fixed header may be followed by header options. After the options, if any, up to 65,535 − 20 − 20 = 65,495 data bytes may follow, where the first 20 refers to the IP header and the second to the TCP header. Segments without any data are legal and are commonly used for acknowledgements and control messages.

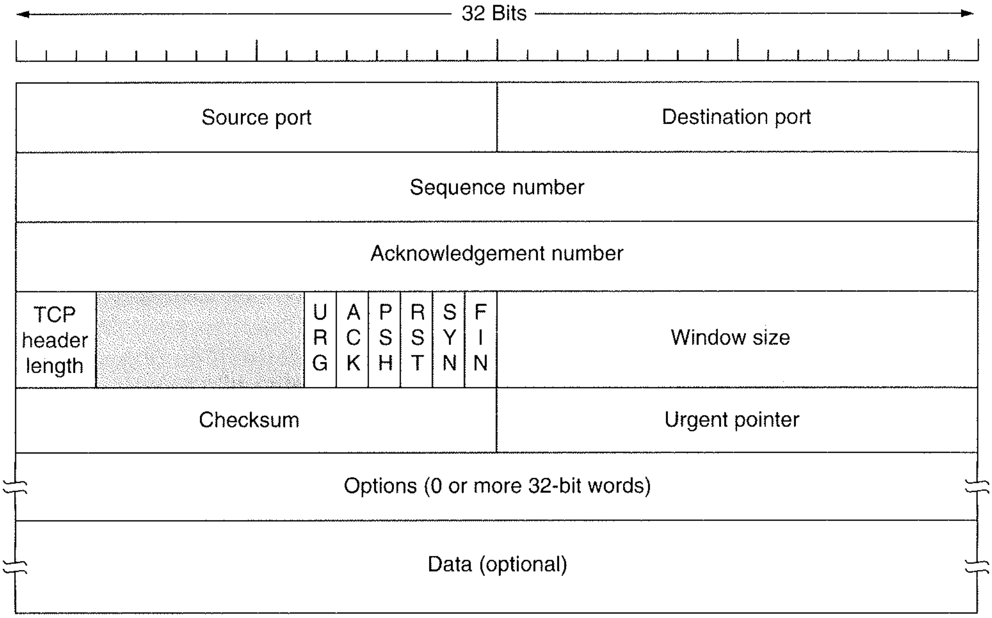


Fig. 6-24. The TCP header.

Let us dissect the TCP header field by field. The *Source port* and *Destination port* fields identify the local end points of the connection. Each host may decide for itself how to allocate its own ports starting at 256. A port plus its host's IP address forms a 48-bit unique TSAP. The source and destination socket numbers together identify the connection.

The *Sequence number* and *Acknowledgement number* fields perform their usual functions. Note that the latter specifies the next byte expected, not the last byte correctly received. Both are 32 bits long because every byte of data is numbered in a TCP stream.

The *TCP header length* tells how many 32-bit words are contained in the TCP header. This information is needed because the *Options* field is of variable length, so the header is too. Technically, this field really indicates the start of the data within the segment, measured in 32-bit words, but that number is just the header length in words, so the effect is the same.

Next comes a 6-bit field that is not used. The fact that this field has survived intact for over a decade is testimony to how well thought out TCP is. Lesser protocols would have needed it to fix bugs in the original design.

Now come six 1-bit flags. *URG* is set to 1 if the *Urgent pointer* is in use. The *Urgent pointer* is used to indicate a byte offset from the current sequence number at which urgent data are to be found. This facility is in lieu of interrupt messages. As we mentioned above, this facility is a bare bones way of allowing the sender to signal the receiver without getting TCP itself involved in the reason for the interrupt.

The *ACK* bit is set to 1 to indicate that the *Acknowledgement number* is valid. If *ACK* is 0, the segment does not contain an acknowledgement so the *Acknowledgement number* field is ignored.

The *PSH* bit indicates PUSHed data. The receiver is hereby kindly requested to deliver the data to the application upon arrival and not buffer it until a full buffer has been received (which it might otherwise do for efficiency reasons).

The *RST* bit is used to reset a connection that has become confused due to a host crash or some other reason. It is also used to reject an invalid segment or refuse an attempt to open a connection. In general, if you get a segment with the *RST* bit on, you have a problem on your hands.

The *SYN* bit is used to establish connections. The connection request has $SYN = 1$ and $ACK = 0$ to indicate that the piggyback acknowledgement field is not in use. The connection reply does bear an acknowledgement, so it has $SYN = 1$ and $ACK = 1$. In essence the *SYN* bit is used to denote CONNECTION REQUEST and CONNECTION ACCEPTED, with the *ACK* bit used to distinguish between those two possibilities.

The *FIN* bit is used to release a connection. It specifies that the sender has no more data to transmit. However, after closing a connection, a process may continue to receive data indefinitely. Both *SYN* and *FIN* segments have sequence numbers and are thus guaranteed to be processed in the correct order.

Flow control in TCP is handled using a variable-size sliding window. The *Window size* field tells how many bytes may be sent starting at the byte acknowledged. A *Window size* field of 0 is legal and says that the bytes up to and including *Acknowledgement number* − 1 have been received, but that the receiver is currently badly in need of a rest and would like no more data for the moment, thank you. Permission to send can be granted later by sending a segment with the same *Acknowledgement number* and a nonzero *Window size* field.

A *Checksum* is also provided for extreme reliability. It checksums the header, the data, and the conceptual pseudoheader shown in Fig. 6-25. When performing this computation, the TCP *Checksum* field is set to zero, and the data field is padded out with an additional zero byte if its length is an odd number. The checksum algorithm is simply to add up all the 16-bit words in 1's complement and then to take the 1's complement of the sum. As a consequence, when the receiver performs the calculation on the entire segment, including the *Checksum* field, the result should be 0.

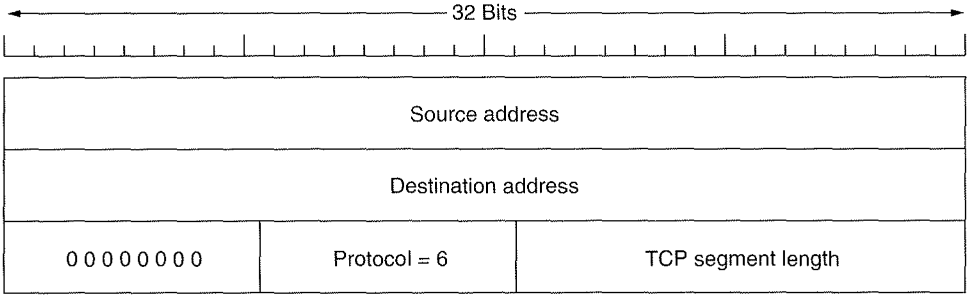


Fig. 6-25. The pseudoheader included in the TCP checksum.

The pseudoheader contains the 32-bit IP addresses of the source and destination machines, the protocol number for TCP (6), and the byte count for the TCP segment (including the header). Including the pseudoheader in the TCP checksum computation helps detect misdelivered packets, but doing so violates the protocol hierarchy since the IP addresses in it belong to the IP layer, not the TCP layer.

The *Options* field was designed to provide a way to add extra facilities not covered by the regular header. The most important option is the one that allows each host to specify the maximum TCP payload it is willing to accept. Using large segments is more efficient than using small ones because the 20-byte header can then be amortized over more data, but small hosts may not be able to handle very large segments. During connection setup, each side can announce its maximum and see its partner's. The smaller of the two numbers wins. If a host does not use this option, it defaults to a 536-byte payload. All Internet hosts are required to accept TCP segments of 536 + 20 = 556 bytes.

For lines with high bandwidth, high delay, or both, the 64 KB window is often a problem. On a T3 line (44.736 Mbps), it takes only 12 msec to output a full 64 KB window. If the round trip propagation delay is 50 msec (typical for a transcontinental fiber), the sender will be idle 3/4 of the time waiting for acknowledgements. On a satellite connection, the situation is even worse. A larger window size would allow the sender to keep pumping data out, but using the 16-bit *Window size* field, there is no way to express such a size. In RFC 1323, a *Window scale* option was proposed, allowing the sender and receiver to negotiate a window scale factor. This number allows both sides to shift the *Window size* field up to 16 bits to the left, thus allowing windows of up to 2^{32} bytes. Most TCP implementations now support this option.

Another option proposed by RFC 1106 and now widely implemented is the use of the selective repeat instead of go back n protocol. If the receiver gets one bad segment and then a large number of good ones, the normal TCP protocol will eventually time out and retransmit all the unacknowledged segments, including all those that were received correctly. RFC 1106 introduced NAKs, to allow the receiver to ask for a specific segment (or segments). After it gets these, it can acknowledge all the buffered data, thus reducing the amount of data retransmitted.

6.4.4. TCP Connection Management

Connections are established in TCP using the three-way handshake discussed in Sec. 6.2.2. To establish a connection, one side, say the server, passively waits for an incoming connection by executing the LISTEN and ACCEPT primitives, either specifying a specific source or nobody in particular.

The other side, say the client, executes a CONNECT primitive, specifying the IP address and port to which it wants to connect, the maximum TCP segment size it is willing to accept, and optionally some user data (e.g., a password). The CONNECT primitive sends a TCP segment with the *SYN* bit on and *ACK* bit off and waits for a response.

When this segment arrives at the destination, the TCP entity there checks to see if there is a process that has done a LISTEN on the port given in the *Destination port* field. If not, it sends a reply with the *RST* bit on to reject the connection.

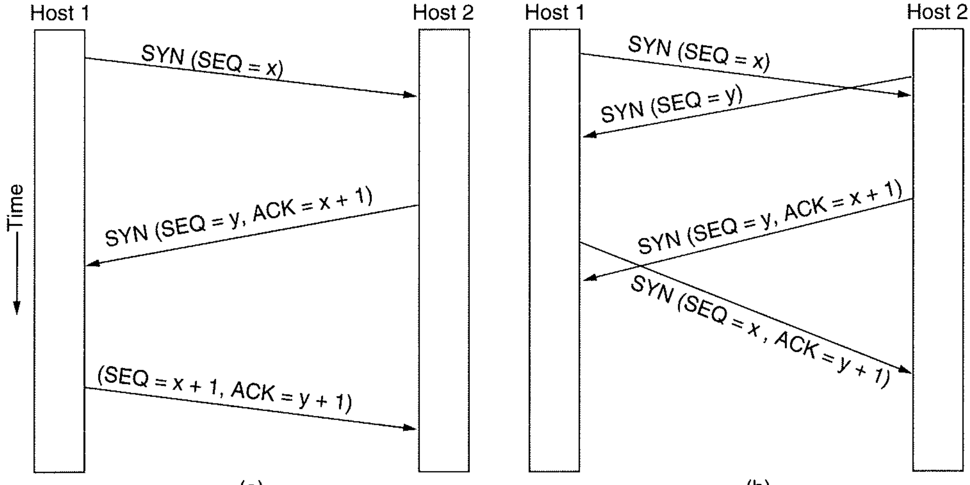


Fig. 6-26. (a) TCP connection establishment in the normal case. (b) Call collision.

If some process is listening to the port, that process is given the incoming TCP segment. It can then either accept or reject the connection. If it accepts, an acknowledgement segment is sent back. The sequence of TCP segments sent in the normal case is shown in Fig. 6-26(a). Note that a *SYN* segment consumes 1 byte of sequence space so it can be acknowledged unambiguously.

In the event that two hosts simultaneously attempt to establish a connection between the same two sockets, the sequence of events is as illustrated in Fig. 6-26(b). The result of these events is that just one connection is established, not two because connections are identified by their end points. If the first setup results in a connection identified by (x, y) and the second one does too, only one table entry is made, namely, for (x, y).

The initial sequence number on a connection is not 0 for the reasons we discussed earlier. A clock-based scheme is used, with a clock tick every 4 µsec. For additional safety, when a host crashes, it may not reboot for the maximum packet lifetime (120 sec) to make sure that no packets from previous connections are still roaming around the Internet somewhere.

Although TCP connections are full duplex, to understand how connections are released it is best to think of them as a pair of simplex connections. Each simplex connection is released independently of its sibling. To release a connection, either party can send a TCP segment with the *FIN* bit set, which means that it has no more data to transmit. When the *FIN* is acknowledged, that direction is shut down for new data. Data may continue to flow indefinitely in the other direction, however. When both directions have been shut down, the connection is released. Normally, four TCP segments are needed to release a connection, one *FIN* and one *ACK* for each direction. However, it is possible for the first *ACK* and the second *FIN* to be contained in the same segment, reducing the total count to three.

Just as with telephone calls in which both people say goodbye and hang up the phone simultaneously, both ends of a TCP connection may send *FIN* segments at the same time. These are each acknowledged in the usual way, and the connection shut down. There is, in fact, no essential difference between the two hosts releasing sequentially or simultaneously.

To avoid the two-army problem, timers are used. If a response to a *FIN* is not forthcoming within two maximum packet lifetimes, the sender of the *FIN* releases the connection. The other side will eventually notice that nobody seems to be listening to it any more, and time out as well. While this solution is not perfect, given the fact that a perfect solution is theoretically impossible, it will have to do. In practice, problems rarely arise.

The steps required to establish and release connections can be represented in a finite state machine with the 11 states listed in Fig. 6-27. In each state, certain events are legal. When a legal event happens, some action may be taken. If some other event happens, an error is reported.

Each connection starts in the *CLOSED* state. It leaves that state when it does either a passive open (LISTEN), or an active open (CONNECT). If the other side does the opposite one, a connection is established and the state becomes *ESTABLISHED*. Connection release can be initiated by either side. When it is complete, the state returns to *CLOSED*.

The finite state machine itself is shown in Fig. 6-28. The common case of a client actively connecting to a passive server is shown with heavy lines—solid for

State	Description
CLOSED	No connection is active or pending
LISTEN	The server is waiting for an incoming call
SYN RCVD	A connection request has arrived; wait for ACK
SYN SENT	The application has started to open a connection
ESTABLISHED	The normal data transfer state
FIN WAIT 1	The application has said it is finished
FIN WAIT 2	The other side has agreed to release
TIMED WAIT	Wait for all packets to die off
CLOSING	Both sides have tried to close simultaneously
CLOSE WAIT	The other side has initiated a release
LAST ACK	Wait for all packets to die off

Fig. 6-27. The states used in the TCP connection management finite state machine.

the client, dotted for the server. The lightface lines are unusual event sequences. Each line in Fig. 6-28 is marked by an *event/action* pair. The event can either be a user-initiated system call (CONNECT, LISTEN, SEND, or CLOSE), a segment arrival (*SYN*, *FIN*, *ACK*, or *RST*), or in one case, a timeout of twice the maximum packet lifetime. The action is the sending of a control segment (*SYN*, *FIN*, or *RST*) or nothing, indicated by —. Comments are shown in parentheses.

The diagram can best be understood by first following the path of a client (the heavy solid line) then later the path of a server (the heavy dashed line). When an application on the client machine issues a CONNECT request, the local TCP entity creates a connection record, marks it as being in the *SYN SENT* state, and sends a *SYN* segment. Note that many connections may be open (or being opened) at the same time on behalf of multiple applications, so the state is per connection and recorded in the connection record. When the *SYN+ACK* arrives, TCP sends the final *ACK* of the three-way handshake and switches into the *ESTABLISHED* state. Data can now be sent and received.

When an application is finished, it executes a CLOSE primitive, which causes the local TCP entity to send a *FIN* segment and wait for the corresponding *ACK* (dashed box marked active close). When the *ACK* arrives, a transition is made to state *FIN WAIT 2* and one direction of the connection is now closed. When the other side closes, too, a *FIN* comes in, which is acknowledged. Now both sides are closed, but TCP waits a time equal to the maximum packet lifetime to guarantee that all packets from the connection have died off, just in case the acknowledgement was lost. When the timer goes off, TCP deletes the connection record.

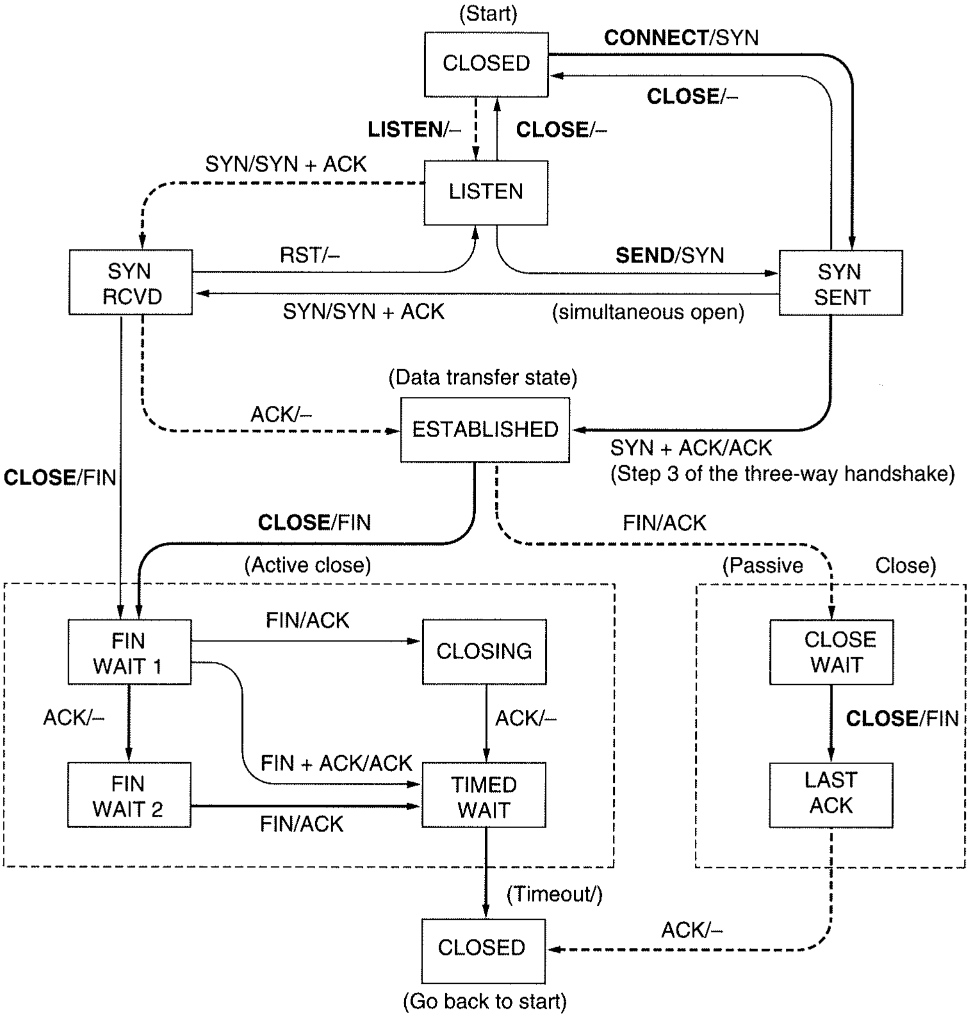


Fig. 6-28. TCP connection management finite state machine. The heavy solid line is the normal path for a client. The heavy dashed line is the normal path for a server. The light lines are unusual events.

Now let us examine connection management from the server's viewpoint. The server does a LISTEN and settles down to see who turns up. When a *SYN* comes in, it is acknowledged and the server goes to the *SYN RCVD* state. When the server's *SYN* is itself acknowledged, the three-way handshake is complete and the server goes to the *ESTABLISHED* state. Data transfer can now occur.

When the client has had enough, it does a CLOSE, which causes a *FIN* to arrive at the server (dashed box marked passive close). The server is then signaled. When it, too, does a CLOSE, a *FIN* is sent to the client. When the client's acknowledgement shows up, the server releases the connection and deletes the connection record.

6.4.5. TCP Transmission Policy

Window management in TCP is not directly tied to acknowledgements as it is in most data link protocols. For example, suppose the receiver has a 4096-byte buffer as shown in Fig. 6-29. If the sender transmits a 2048-byte segment that is correctly received, the receiver will acknowledge the segment. However, since it now has only 2048 of buffer space (until the application removes some data from the buffer), it will advertise a window of 2048 starting at the next byte expected.

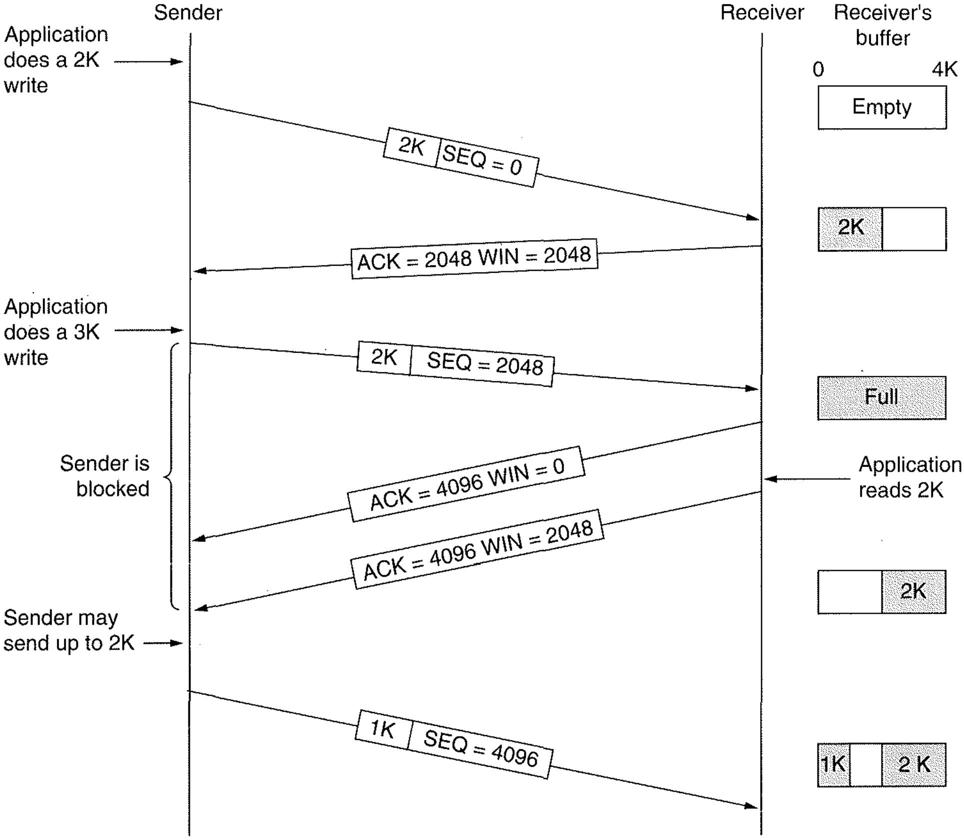


Fig. 6-29. Window management in TCP.

Now the sender transmits another 2048 bytes, which are acknowledged, but the advertised window is 0. The sender must stop until the application process on the receiving host has removed some data from the buffer, at which time TCP can advertise a larger window.

When the window is 0, the sender may not normally send segments, with two exceptions. First, urgent data may be sent, for example, to allow the user to kill the process running on the remote machine. Second, the sender may send a 1-byte segment to make the receiver reannounce the next byte expected and window size. The TCP standard explicitly provides this option to prevent deadlock if a window announcement ever gets lost.

Senders are not required to transmit data as soon as they come in from the application. Neither are receivers required to send acknowledgements as soon as possible. For example, in Fig. 6-29, when the first 2 KB of data came in, TCP, knowing that it had a 4-KB window available, would have been completely correct in just buffering the data until another 2 KB came in, to be able to transmit a segment with a 4-KB payload. This freedom can be exploited to improve performance.

Consider a TELNET connection to an interactive editor that reacts on every keystroke. In the worst case, when a character arrives at the sending TCP entity, TCP creates a 21-byte TCP segment, which it gives to IP to send as a 41-byte IP datagram. At the receiving side, TCP immediately sends a 40-byte acknowledgement (20 bytes of TCP header and 20 bytes of IP header). Later, when the editor has read the byte, TCP sends a window update, moving the window 1 byte to the right. This packet is also 40 bytes. Finally, when the editor has processed the character, it echoes it as a 41-byte packet. In all, 162 bytes of bandwidth are used and four segments are sent for each character typed. When bandwidth is scarce, this method of doing business is not desirable.

One approach that many TCP implementations use to optimize this situation is to delay acknowledgements and window updates for 500 msec in the hope of acquiring some data on which to hitch a free ride. Assuming the editor echoes within 500 msec, only one 41-byte packet now need be sent back to the remote user, cutting the packet count and bandwidth usage in half.

Although this rule reduces the load placed on the network by the receiver, the sender is still operating inefficiently by sending 41-byte packets containing 1 byte of data. A way to reduce this usage is known as **Nagle's algorithm** (Nagle, 1984). What Nagle suggested is simple: when data come into the sender one byte at a time, just send the first byte and buffer all the rest until the outstanding byte is acknowledged. Then send all the buffered characters in one TCP segment and start buffering again until they are all acknowledged. If the user is typing quickly and the network is slow, a substantial number of characters may go in each segment, greatly reducing the bandwidth used. The algorithm additionally allows a new packet to be sent if enough data have trickled in to fill half the window or a maximum segment.

Nagle's algorithm is widely used by TCP implementations, but there are times when it is better to disable it. In particular, when an X-Windows application is being run over the Internet, mouse movements have to be sent to the remote computer. Gathering them up to send in bursts makes the mouse cursor move erratically, which makes for unhappy users.

Another problem that can ruin TCP performance is the **silly window syndrome** (Clark, 1982). This problem occurs when data are passed to the sending TCP entity in large blocks, but an interactive application on the receiving side reads data 1 byte at a time. To see the problem, look at Fig. 6-30. Initially, the TCP buffer on the receiving side is full and the sender knows this (i.e., has a window of size 0). Then the interactive application reads one character from the TCP stream. This action makes the receiving TCP happy, so it sends a window update to the sender saying that it is all right to send 1 byte. The sender obliges and sends 1 byte. The buffer is now full, so the receiver acknowledges the 1-byte segment but sets the window to 0. This behavior can go on forever.

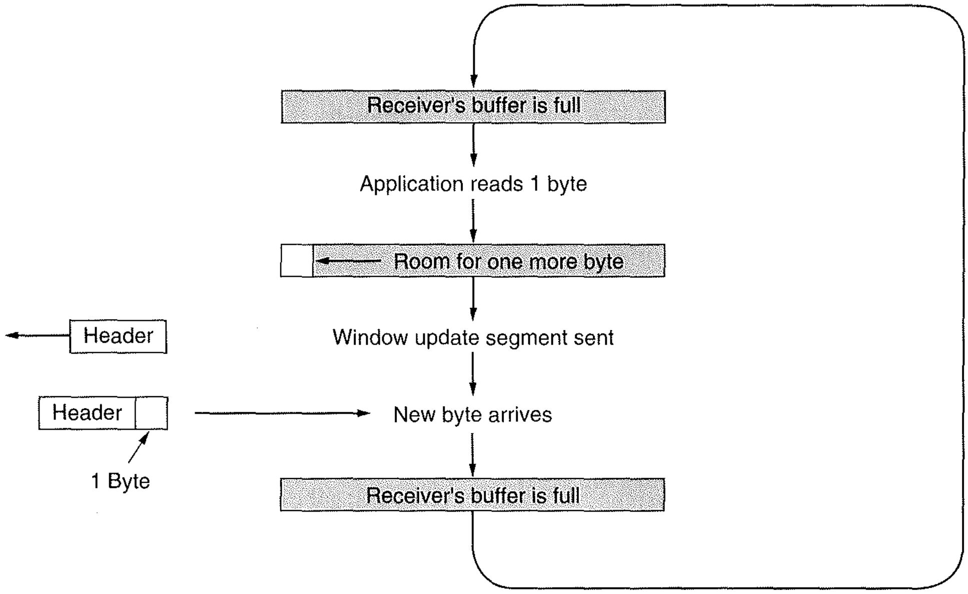


Fig. 6-30. Silly window syndrome.

Clark's solution is to prevent the receiver from sending a window update for 1 byte. Instead it is forced to wait until it has a decent amount of space available and advertise that instead. Specifically, the receiver should not send a window update until it can handle the maximum segment size it advertised when the connection was established, or its buffer is half empty, whichever is smaller.

Furthermore, the sender can also help by not sending tiny segments. Instead, it should try to wait until it has accumulated enough space in the window to send a full segment or at least one containing half of the receiver's buffer size (which it must estimate from the pattern of window updates it has received in the past).

Nagle's algorithm and Clark's solution to the silly window syndrome are complementary. Nagle was trying to solve the problem caused by the sending application delivering data to TCP a byte at a time. Clark was trying to solve the problem of the receiving application sucking the data up from TCP a byte at a time. Both solutions are valid and can work together. The goal is for the sender not to send small segments and the receiver not to ask for them.

The receiving TCP can go further in improving performance than just doing window updates in large units. Like the sending TCP, it also has the ability to buffer data, so it can block a READ request from the application until it has a large chunk of data to provide. Doing this reduces the number of calls to TCP, and hence the overhead. Of course, it also increases the response time, but for noninteractive applications like file transfer, efficiency may outweigh response time to individual requests.

Another receiver issue is what to do with out of order segments. They can be kept or discarded, at the receiver's discretion. Of course, acknowledgements can be sent only when all the data up to the byte acknowledged have been received. If the receiver gets segments 0, 1, 2, 4, 5, 6, and 7, it can acknowledge everything up to and including the last byte in segment 2. When the sender times out, it then retransmits segment 3. If the receiver has buffered segments 4 through 7, upon receipt of segment 3 it can acknowledge all bytes up to the end of segment 7.

6.4.6. TCP Congestion Control

When the load offered to any network is more than it can handle, congestion builds up. The Internet is no exception. In this section we will discuss algorithms that have been developed over the past decade to deal with congestion. Although the network layer also tries to manage congestion, most of the heavy lifting is done by TCP because the real solution to congestion is to slow down the data rate.

In theory, congestion can be dealt with by employing a principle borrowed from physics: the law of conservation of packets. The idea is not to inject a new packet into the network until an old one leaves (i.e., is delivered). TCP attempts to achieve this goal by dynamically manipulating the window size.

The first step in managing congestion is detecting it. In the old days, detecting congestion was difficult. A timeout caused by a lost packet could have been caused by either (1) noise on a transmission line or (2) packet discard at a congested router. Telling the difference was difficult.

Nowadays, packet loss due to transmission errors is relatively rare because most long-haul trunks are fiber (although wireless networks are a different story). Consequently, most transmission timeouts on the Internet are due to congestion. All the Internet TCP algorithms assume that timeouts are caused by congestion and monitor timeouts for signs of trouble the way miners watch their canaries.

Before discussing how TCP reacts to congestion, let us first describe what it does to try to prevent it from occurring in the first place. When a connection is established, a suitable window size has to be chosen. The receiver can specify a window based on its buffer size. If the sender sticks to this window size, problems will not occur due to buffer overflow at the receiving end, but they may still occur due to internal congestion within the network.

In Fig. 6-31, we see this problem illustrated hydraulically. In Fig. 6-31(a), we see a thick pipe leading to a small-capacity receiver. As long as the sender does not send more water than the bucket can contain, no water will be lost. In Fig. 6-31(b), the limiting factor is not the bucket capacity, but the internal carrying capacity of the network. If too much water comes in too fast, it will back up and some will be lost (in this case by overflowing the funnel).

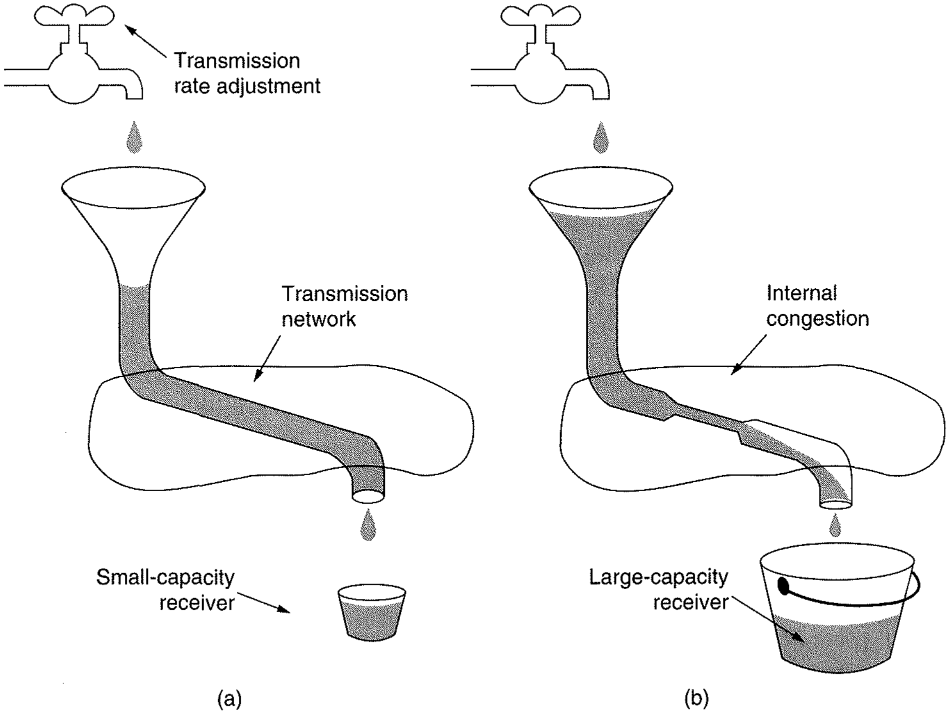


Fig. 6-31. (a) A fast network feeding a low-capacity receiver. (b) A slow network feeding a high-capacity receiver.

The Internet solution is to realize that two potential problems exist—network capacity and receiver capacity—and to deal with each of them separately. To do so, each sender maintains two windows: the window the receiver has granted and a second window, the **congestion window**. Each reflects the number of bytes the sender may transmit. The number of bytes that may be sent is the minimum of the two windows. Thus the effective window is the minimum of what the sender thinks is all right and what the receiver thinks is all right. If the receiver says "Send 8K" but the sender knows that bursts of more than 4K clog the network up, it sends 4K. On the other hand, if the receiver says "Send 8K" and the sender knows that bursts of up to 32K get through effortlessly, it sends the full 8K requested.

When a connection is established, the sender initializes the congestion window to the size of the maximum segment in use on the connection. It then sends one maximum segment. If this segment is acknowledged before the timer goes off, it adds one segment's worth of bytes to the congestion window to make it two maximum size segments and sends two segments. As each of these segments is acknowledged, the congestion window is increased by one maximum segment size. When the congestion window is *n* segments, if all *n* are acknowledged on time, the congestion window is increased by the byte count corresponding to *n* segments. In effect, each burst successfully acknowledged doubles the congestion window.

The congestion window keeps growing exponentially until either a timeout occurs or the receiver's window is reached. The idea is that if bursts of size, say, 1024, 2048, and 4096 bytes work fine, but a burst of 8192 bytes gives a timeout, the congestion window should be set to 4096 to avoid congestion. As long as the congestion window remains at 4096, no bursts longer than that will be sent, no matter how much window space the receiver grants. This algorithm is called **slow start**, but it is not slow at all (Jacobson, 1988). It is exponential. All TCP implementations are required to support it.

Now let us look at the Internet congestion control algorithm. It uses a third parameter, the **threshold**, initially 64K, in addition to the receiver and congestion windows. When a timeout occurs, the threshold is set to half of the current congestion window, and the congestion window is reset to one maximum segment. Slow start is then used to determine what the network can handle, except that exponential growth stops when the threshold is hit. From that point on, successful transmissions grow the congestion window linearly (by one maximum segment for each burst) instead of one per segment. In effect, this algorithm is guessing that it is probably acceptable to cut the congestion window in half, and then it gradually works its way up from there.

As an illustration of how the congestion algorithm works, see Fig. 6-32. The maximum segment size here is 1024 bytes. Initially the congestion window was 64K, but a timeout occurred, so the threshold is set to 32K and the congestion window to 1K for transmission 0 here. The congestion window then grows exponentially until it hits the threshold (32K). Starting then it grows linearly.

Transmission 13 is unlucky (it should have known) and a timeout occurs. The threshold is set to half the current window (by now 40K, so half is 20K) and slow start initiated all over again. When the acknowledgements from transmission 14 start coming in, the first four each double the congestion window, but after that, growth becomes linear again.

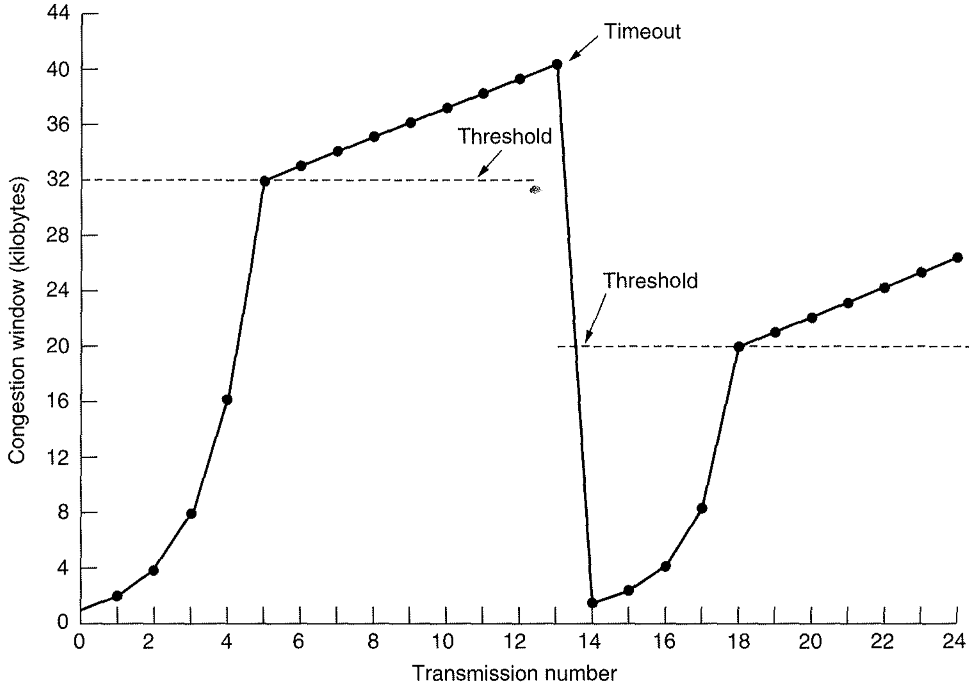


Fig. 6-32. An example of the Internet congestion algorithm.

If no more timeouts occur, the congestion window will continue to grow up to the size of the receiver's window. At that point, it will stop growing and remain constant as long as there are no more timeouts and the receiver's window does not change size. As an aside, if an ICMP SOURCE QUENCH packet comes in and is passed to TCP, this event is treated the same way as a timeout.

Work on improving the congestion control mechanism is continuing. For example, Brakmo et al. (1994) have reported improving TCP throughput by 40 percent to 70 percent by managing the clock more accurately, predicting congestion before timeouts occur, and using this early warning system to improve the slow start algorithm.

6.4.7. TCP Timer Management

TCP uses multiple timers (at least conceptually) to do its work. The most important of these is the **retransmission timer**. When a segment is sent, a retransmission timer is started. If the segment is acknowledged before the timer expires, the timer is stopped. If, on the other hand, the timer goes off before the acknowledgement comes in, the segment is retransmitted (and the timer started again). The question that arises is: How long should the timeout interval be?

This problem is much more difficult in the Internet transport layer than in the generic data link protocols of Chap. 3. In the latter case, the expected delay is highly predictable (i.e., has a low variance), so the timer can be set to go off just slightly after the acknowledgement is expected, as shown in Fig. 6-33(a). Since acknowledgements are rarely delayed in the data link layer, the absence of an acknowledgement at the expected time generally means the frame or the acknowledgement has been lost.

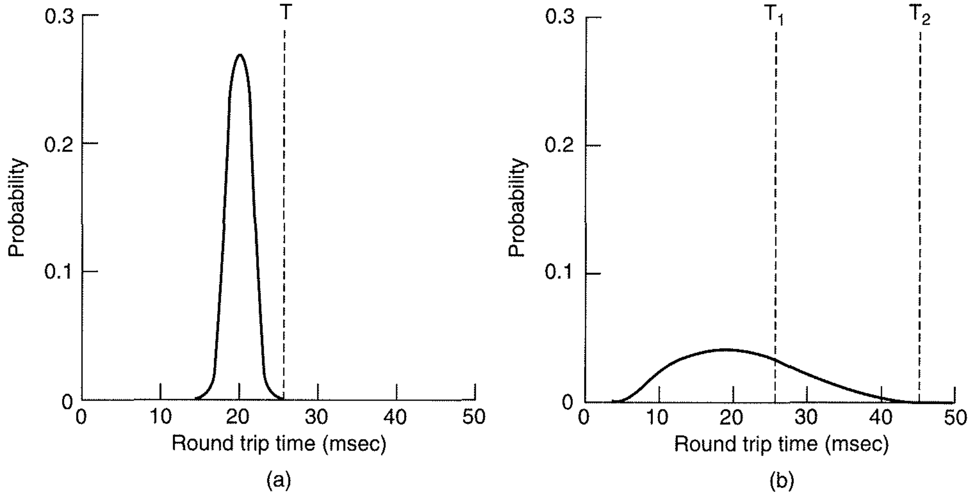


Fig. 6-33. (a) Probability density of acknowledgement arrival times in the data link layer. (b) Probability density of acknowledgement arrival times for TCP.

TCP is faced with a radically different environment. The probability density function for the time it takes for a TCP acknowledgement to come back looks more like Fig. 6-33(b) than Fig. 6-33(a). Determining the round-trip time to the destination is tricky. Even when it is known, deciding on the timeout interval is also difficult. If the timeout is set too short, say T_1 in Fig. 6-33(b), unnecessary retransmissions will occur, clogging the Internet with useless packets. If it is set too long, (T_2), performance will suffer due to the long retransmission delay whenever a packet is lost. Furthermore, the mean and variance of the acknowledgement arrival distribution can change rapidly within a few seconds as congestion builds up or is resolved.

The solution is to use a highly dynamic algorithm that constantly adjusts the timeout interval, based on continuous measurements of network performance. The algorithm generally used by TCP is due to Jacobson (1988) and works as follows. For each connection, TCP maintains a variable, *RTT*, that is the best current estimate of the round-trip time to the destination in question. When a segment is sent, a timer is started, both to see how long the acknowledgement takes and to trigger a retransmission if it takes too long. If the acknowledgement gets back before the timer expires, TCP measures how long the acknowledgement took, say, M. It then updates RTT according to the formula

$$RTT = \alpha RTT + (1 - \alpha)M$$

where α is a smoothing factor that determines how much weight is given to the old value. Typically $\alpha = 7/8$.

Even given a good value of RTT, choosing a suitable retransmission timeout is a nontrivial matter. Normally, TCP uses βRTT, but the trick is choosing β. In the initial implementations, β was always 2, but experience showed that a constant value was inflexible because it failed to respond when the variance went up.

In 1988, Jacobson proposed making β roughly proportional to the standard deviation of the acknowledgement arrival time probability density function so a large variance means a large β and vice versa. In particular, he suggested using the *mean deviation* as a cheap estimator of the *standard deviation*. His algorithm requires keeping track of another smoothed variable, D, the deviation. Whenever an acknowledgement comes in, the difference between the expected and observed values, $|RTT - M|$ is computed. A smoothed value of this is maintained in D by the formula

$$D = \alpha D + (1 - \alpha)\ |RTT - M|$$

where α may or may not be the same value used to smooth RTT. While D is not exactly the same as the standard deviation, it is good enough and Jacobson showed how it could be computed using only integer adds, subtracts, and shifts, a big plus. Most TCP implementations now use this algorithm and set the timeout interval to

$$\text{Timeout} = RTT + 4 * D$$

The choice of the factor 4 is somewhat arbitrary, but it has two advantages. First, multiplication by 4 can be done with a single shift. Second, it minimizes unnecessary timeouts and retransmissions because less than one percent of all packets come in more than four standard deviations late. (Actually, Jacobson initially said to use 2, but later work has shown that 4 gives better performance.)

One problem that occurs with the dynamic estimation of RTT is what to do when a segment times out and is sent again. When the acknowledgement comes in, it is unclear whether the acknowledgement refers to the first transmission or a later one. Guessing wrong can seriously contaminate the estimate of RTT. Phil Karn discovered this problem the hard way. He is an amateur radio enthusiast interested in transmitting TCP/IP packets by ham radio, a notoriously unreliable medium (on a good day, half the packets get through). He made a simple proposal: do not update RTT on any segments that have been retransmitted. Instead, the timeout is doubled on each failure until the segments get through the first time. This fix is called **Karn's algorithm**. Most TCP implementations use it.

The retransmission timer is not the only one TCP uses. A second timer is the **persistence timer**. It is designed to prevent the following deadlock. The receiver sends an acknowledgement with a window size of 0, telling the sender to wait. Later, the receiver updates the window, but the packet with the update is lost. Now both the sender and the receiver are waiting for each other to do something. When the persistence timer goes off, the sender transmits a probe to the receiver. The response to the probe gives the window size. If it is still zero, the persistence timer is set again and the cycle repeats. If it is nonzero, data can now be sent.

A third timer that some implementations use is the **keepalive timer**. When a connection has been idle for a long time, the keepalive timer may go off to cause one side to check if the other side is still there. If it fails to respond, the connection is terminated. This feature is controversial because it adds overhead and may terminate an otherwise healthy connection due to a transient network partition.

The last timer used on each TCP connection is the one used in the *TIMED WAIT* state while closing. It runs for twice the maximum packet lifetime to make sure that when a connection is closed, all packets created by it have died off.

6.4.8. UDP

The Internet protocol suite also supports a connectionless transport protocol, **UDP (User Data Protocol)**. UDP provides a way for applications to send encapsulated raw IP datagrams and send them without having to establish a connection. Many client-server applications that have one request and one response use UDP rather than go to the trouble of establishing and later releasing a connection. UDP is described in RFC 768.

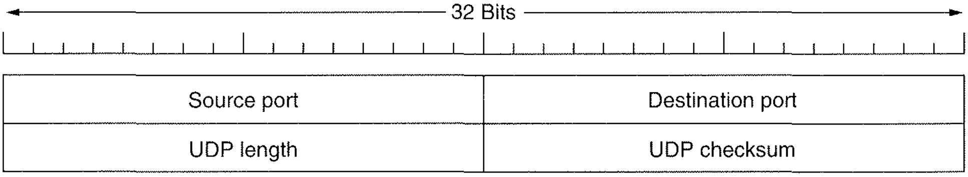


Fig. 6-34. The UDP header.

A UDP segment consists of an 8-byte header followed by the data. The header is shown in Fig. 6-34. The two ports serve the same function as they do in TCP: to identify the end points within the source and destination machines. The *UDP length* field includes the 8-byte header and the data. The *UDP checksum* includes the same format pseudoheader shown in Fig. 6-25, the UDP header, and the UDP data, padded out to an even number of bytes if need be. It is optional and stored as 0 if not computed (a true computed 0 is stored as all 1s, which is the same in 1's complement). Turning it off is foolish unless the quality of the data does not matter (e.g., digitized speech).

6.4.9. Wireless TCP and UDP

In theory, transport protocols should be independent of the technology of the underlying network layer. In particular, TCP should not care whether IP is running over fiber or over radio. In practice, it does matter because most TCP implementations have been carefully optimized based on assumptions that are true for wired networks but which fail for wireless networks. Ignoring the properties of wireless transmission can lead to a TCP implementation that is logically correct but has horrendous performance.

The principal problem is the congestion control algorithm. Nearly all TCP implementations nowadays assume that timeouts are caused by congestion, not by lost packets. Consequently, when a timer goes off, TCP slows down and sends less vigorously (e.g., Jacobson's slow start algorithm). The idea behind this approach is to reduce the network load and thus alleviate the congestion.

Unfortunately, wireless transmission links are highly unreliable. They lose packets all the time. The proper approach to dealing with lost packets is to send them again, and as quickly as possible. Slowing down just makes matters worse. If, say, 20 percent of all packets are lost, then when the sender transmits 100 packets/sec, the throughput is 80 packets/sec. If the sender slows down to 50 packets/sec, the throughput drops to 40 packets/sec.

In effect, when a packet is lost on a wired network, the sender should slow down. When one is lost on a wireless network, the sender should try harder. When the sender does not know what the network is, it is difficult to make the correct decision.

Frequently, the path from sender to receiver is inhomogeneous. The first 1000 km might be over a wired network, but the last 1 km might be wireless. Now making the correct decision on a timeout is even harder, since it matters where the problem occurred. A solution proposed by Bakne and Badrinath (1995), **indirect TCP**, is to split the TCP connection into two separate connections, as shown in Fig. 6-35. The first connection goes from the sender to the base station. The second one goes from the base station to the receiver. The base station simply copies packets between the connections in both directions.

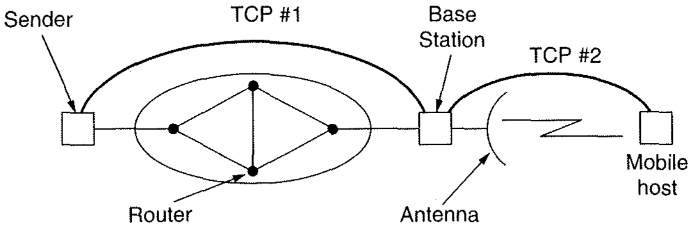


Fig. 6-35. Splitting a TCP connection into two connections.

The advantage of this scheme is that both connections are now homogeneous. Timeouts on the first connection can slow the sender down, whereas timeouts on the second one can speed it up. Other parameters can also be tuned separately for the two connections. The disadvantage is that it violates the semantics of TCP. Since each part of the connection is a full TCP connection, the base station acknowledges each TCP segment in the usual way. Only now, receipt of an acknowledgement by the sender does not mean that the receiver got the segment, only that the base station got it.

A different solution, due to Balakrishnan et al. (1995), does not break the semantics of TCP. It works by making several small modifications to the network layer code in the base station. One of the changes is the addition of a snooping agent that observes and caches TCP segments going out to the mobile host, and acknowledgements coming back from it. When the snooping agent sees a TCP segment going out to the mobile host but does not see an acknowledgement coming back before its (relatively short) timer goes off, it just retransmits that segment, without telling the source that it is doing so. It also generates a retransmission when it sees duplicate acknowledgements from the mobile host go by, invariably meaning that the mobile host has missed something. Duplicate acknowledgements are discarded on the spot, to avoid having the source misinterpret them as a sign of congestion.

One disadvantage of this transparency, however, is that if the wireless link is very lossy, the source may time out waiting for an acknowledgement and invoke the congestion control algorithm. With indirect TCP, the congestion control algorithm will never be started unless there really is congestion in the wired part of the network.

The Balakrishnan et al. paper also has a solution to the problem of lost segments originating at the mobile host. When the base station notices a gap in the inbound sequence numbers, it generates a request for a selective repeat of the missing bytes using a TCP option. Using these two fixes, the wireless link is made more reliable in both directions, without the source knowing about it, and without changing the semantics of TCP.

While UDP does not suffer from the same problems as TCP, wireless communication also introduces difficulties for it. The main trouble is that programs use UDP expecting it to be highly reliable. They know that no guarantees are given, but they still expect it to be near perfect. In a wireless environment, it will be far from perfect. For programs that are able to recover from lost UDP messages, but only at considerable cost, suddenly going from an environment where messages theoretically can be lost but rarely are, to one in which they are constantly being lost can result in a performance disaster.

Wireless communication also affects areas other than just performance. For example, how does a mobile host find a local printer to connect to, rather than use its home printer? Somewhat related to this is how to get the WWW page for the local cell, even if its name is not known. Also, WWW page designers tend to assume lots of bandwidth is available. Putting a large logo on every page becomes counterproductive if it is going to take 30 sec to transmit at 9600 bps every time the page is referenced, irritating the users no end.

6.5. THE ATM AAL LAYER PROTOCOLS

It is not really clear whether or not ATM has a transport layer. On the one hand, the ATM layer has the functionality of a network layer, and there is another layer on top of it (AAL), which sort of makes AAL a transport layer. Some experts agree with this view (e.g., De Prycker, 1993, page 112). One of the protocols used here (AAL 5) is functionally similar to UDP, which is unquestionably a transport protocol.

On the other hand, none of the AAL protocols provide a reliable end-to-end connection, as TCP does (although with only very minor changes they could). Also, in most applications another transport layer is used on top of AAL. Rather than split hairs, we will discuss the AAL layer and its protocols in this chapter without making a claim that it is a true transport layer.

The AAL layer in ATM networks is radically different than TCP, largely because the designers were primarily interested in transmitting voice and video streams, in which rapid delivery is more important than accurate delivery. Remember that the ATM layer just outputs 53-byte cells one after another. It has no error control, no flow control, and no other control. Consequently, it is not well matched to the requirements that most applications need.

To bridge this gap, in Recommendation I.363, ITU has defined an end-to-end layer on top of the ATM layer. This layer, called **AAL (ATM Adaptation Layer)** has a tortuous history, full of mistakes, revisions, and unfinished business. In the following sections we will look at it and its design.

The goal of AAL is to provide useful services to application programs and to shield them from the mechanics of chopping data up into cells at the source and reassembling them at the destination. When ITU began defining AAL, it realized that different applications had different requirements, so it organized the service space along three axes:

1. Real-time service versus nonreal-time service.
2. Constant bit rate service versus variable bit rate service.
3. Connection-oriented service versus connectionless service.

In principle, with three axes and two values on each axis, eight distinct services can be defined, as shown in Fig. 6-36. ITU felt that only four of these were of any use, and named them classes A, B, C, and D, as noted. The others were not supported. Starting with ATM 4.0, Fig. 6-36 is somewhat obsolete, so it has been presented here mostly as background information to help understand why the

7

THE APPLICATION LAYER

Having finished all the preliminaries, we now come to the application layer, where all the interesting applications can be found. The layers below the application layer are there to provide reliable transport, but they do not do any real work for users. In this chapter we will study some real applications.

However, even in the application layer there is a need for support protocols to allow the real applications to function. Accordingly, we will look at three of these before starting with the applications themselves. The first area is security, which is not a single protocol, but a large number of concepts and protocols that can be used to ensure privacy where needed. The second is DNS, which handles naming within the Internet. The third support protocol is for network management. After that, we will examine four real applications: electronic mail, USENET (net news), the World Wide Web, and finally, multimedia.

domain name server

7.1. NETWORK SECURITY

For the first few decades of their existence, computer networks were primarily used by university researchers for sending email, and by corporate employees for sharing printers. Under these conditions, security did not get a lot of attention. But now, as millions of ordinary citizens are using networks for banking, shopping, and filing their tax returns, network security is looming on the horizon as a potentially massive problem. In the following sections, we will study network security from several angles, point out numerous pitfalls, and discuss many algorithms and protocols for making networks more secure.

Security is a broad topic and covers a multitude of sins. In its simplest form, it is concerned with making sure that nosy people cannot read, or worse yet, modify messages intended for other recipients. It is concerned with people trying to access remote services that they are not authorized to use. It also deals with how to tell whether that message purportedly from the IRS saying: "Pay by Friday or else" is really from the IRS or from the Mafia. Security also deals with the problems of legitimate messages being captured and replayed, and with people trying to deny that they sent certain messages.

Most security problems are intentionally caused by malicious people trying to gain some benefit or harm someone. A few of the most common perpetrators are listed in Fig. 7-1. It should be clear from this list that making a network secure involves a lot more than just keeping it free of programming errors. It involves outsmarting often intelligent, dedicated, and sometimes well-funded adversaries. It should also be clear that measures that will stop casual adversaries will have little impact on the serious ones.

Adversary	Goal
Student	To have fun snooping on people's email
Hacker	To test out someone's security system; steal data
Sales rep	To claim to represent all of Europe, not just Andorra
Businessman	To discover a competitor's strategic marketing plan
Ex-employee	To get revenge for being fired
Accountant	To embezzle money from a company
Stockbroker	To deny a promise made to a customer by email
Con man	To steal credit card numbers for sale
Spy	To learn an enemy's military strength
Terrorist	To steal germ warfare secrets

Fig. 7-1. Some people who cause security problems and why.

Network security problems can be divided roughly into four intertwined areas: secrecy, authentication, nonrepudiation, and integrity control. Secrecy has to do with keeping information out of the hands of unauthorized users. This is what usually comes to mind when people think about network security. Authentication deals with determining whom you are talking to before revealing sensitive information or entering into a business deal. Nonrepudiation deals with signatures:

How do you prove that your customer really placed an electronic order for ten million left-handed doohickeys at 89 cents each when he later claims the price was 69 cents? Finally, how can you be sure that a message you received was really the one sent and not something that a malicious adversary modified in transit or concocted?

All these issues (secrecy, authentication, nonrepudiation, and integrity control) occur in traditional systems, too, but with some significant differences. Secrecy and integrity are achieved by using registered mail and locking documents up. Robbing the mail train is harder than it was in Jesse James' day.

Also, people can usually tell the difference between an original paper document and a photocopy, and it often matters to them. As a test, make a photocopy of a valid check. Try cashing the original check at your bank on Monday. Now try cashing the photocopy of the check on Tuesday. Observe the difference in the bank's behavior. With electronic checks, the original and the copy are indistinguishable. It may take a while for banks to get used to this.

People authenticate other people by recognizing their faces, voices, and handwriting. Proof of signing is handled by signatures on letterhead paper, raised seals, and so on. Tampering can usually be detected by handwriting, paper, and ink experts. None of these options are available electronically. Clearly, other solutions are needed.

Before getting into the solutions themselves, it is worth spending a few moments considering where in the protocol stack network security belongs. There is probably no one single place. Every layer has something to contribute. In the physical layer, wiretapping can be foiled by enclosing transmission lines in sealed tubes containing argon gas at high pressure. Any attempt to drill into a tube will release some gas, reducing the pressure and triggering an alarm. Some military systems use this technique.

In the data link layer, packets on a point-to-point line can be encoded as they leave one machine and decoded as they enter another. All the details can be handled in the data link layer, with higher layers oblivious to what is going on. This solution breaks down when packets have to traverse multiple routers, however, because packets have to be decrypted at each router, leaving them vulnerable to attacks from within the router. Also, it does not allow some sessions to be protected (e.g., those involving on-line purchases by credit card) and others not. Nevertheless, **link encryption**, as this method is called, can be added to any network easily and is often useful.

In the network layer, firewalls can be installed to keep packets in or keep packets out. We looked at firewalls in Chap. 5. In the transport layer, entire connections can be encrypted, end to end, that is, process to process. Although these solutions help with secrecy issues and many people are working hard to improve them, none of them solve the authentication or nonrepudiation problem in a sufficiently general way. To tackle these problems, the solutions must be in the application layer, which is why they are being studied in this chapter.

7.1.1. Traditional Cryptography

Cryptography has a long and colorful history. In this section we will just sketch some of the highlights, as background information for what follows. For a complete history, Kahn's (1967) book is still recommended reading. For a comprehensive treatment of the current state-of-the-art, see (Kaufman et al., 1995; Schneier, 1996; and Stinson, 1995).

Historically, four groups of people have used and contributed to the art of cryptography: the military, the diplomatic corps, diarists, and lovers. Of these, the military has had the most important role and has shaped the field. Within military organizations, the messages to be encrypted have traditionally been given to poorly paid code clerks for encryption and transmission. The sheer volume of messages prevented this work from being done by a few elite specialists.

Until the advent of computers, one of the main constraints on cryptography had been the ability of the code clerk to perform the necessary transformations, often on a battlefield with little equipment. An additional constraint has been the difficulty in switching over quickly from one cryptographic method to another one, since this entails retraining a large number of people. However, the danger of a code clerk being captured by the enemy has made it essential to be able to change the cryptographic method instantly, if need be. These conflicting requirements have given rise to the model of Fig. 7-2.

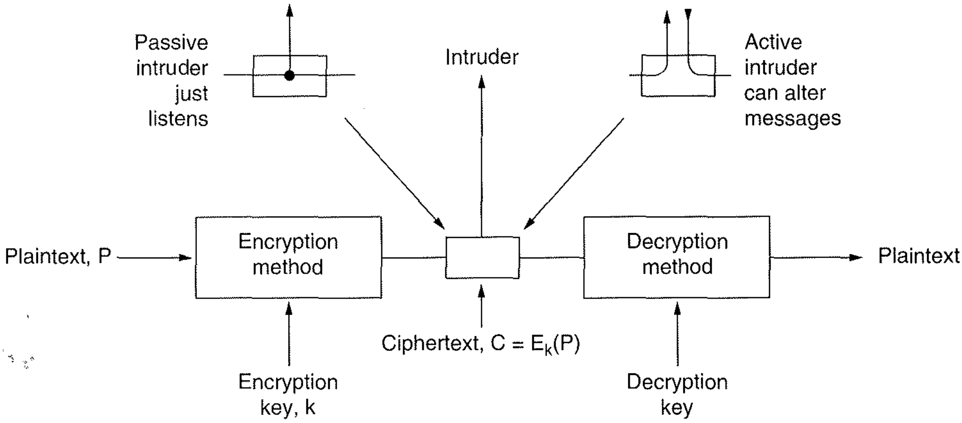


Fig. 7-2. The encryption model.

The messages to be encrypted, known as the **plaintext**, are transformed by a function that is parametrized by a **key**. The output of the encryption process, known as the **ciphertext**, is then transmitted, often by messenger or radio. We assume that the enemy, or **intruder**, hears and accurately copies down the complete ciphertext. However, unlike the intended recipient, he does not know what the decryption key is and so cannot decrypt the ciphertext easily. Sometimes the dollars for the next version, so "A" (Leonard Adleman) was out of luck. Although it has been patched up again, the knapsack algorithm is not considered secure and is rarely used.

Other public-key schemes are based on the difficulty of computing discrete logarithms (Rabin, 1979). Algorithms that use this principle have been invented by El Gamal (1985) and Schnorr (1991).

A few other schemes exist, such as those based on elliptic curves (Menezes and Vanstone, 1993), but the three major categories are those based on the difficulty of factoring large numbers, computing discrete logarithms, and determining the contents of a knapsack from its weight. These problems are thought to be genuinely difficult to solve because mathematicians have been working on them for many years without any great breakthroughs.

7.1.5. Authentication Protocols

Authentication is the technique by which a process verifies that its communication partner is who it is supposed to be and not an imposter. Verifying the identity of a remote process in the face of a malicious, active intruder is surprisingly difficult and requires complex protocols based on cryptography. In this section, we will study some of the many authentication protocols that are used on insecure computer networks.

As an aside, some people confuse authorization with authentication. Authentication deals with the question of whether or not you are actually communicating with a specific process. Authorization is concerned with what that process is permitted to do. For example, a client process contacts a file server and says: "I am Scott's process and I want to delete the file *cookbook.old*." From the file server's point of view, two questions must be answered:

1. Is this actually Scott's process (authentication)?
2. Is Scott allowed to delete *cookbook.old* (authorization)?

Only after both questions have been unambiguously answered in the affirmative can the requested action take place. The former question is really the key one. Once the file server knows whom it is talking to, checking authorization is just a matter of looking up entries in local tables. For this reason, we will concentrate on authentication in this section.

The general model that all authentication protocols use is this. An initiating user (really a process), say, Alice, wants to establish a secure connection with a second user, Bob. Alice and Bob are sometimes called **principals**, the main characters in our story. Bob is a banker with whom Alice would like to do business. Alice starts out by sending a message either to Bob, or to a trusted **key distribution center** (**KDC**), which is always honest. Several other message exchanges follow in various directions. As these message are being sent, a nasty intruder, Trudy,[†] may intercept, modify, or replay them in order to trick Alice and Bob or just to gum up the works.

Nevertheless, when the protocol has been completed, Alice is sure she is talking to Bob and Bob is sure he is talking to Alice. Furthermore, in most of the protocols, the two of them will also have established a secret **session key** for use in the upcoming conversation. In practice, for performance reasons, all data traffic is encrypted using secret-key cryptography, although public-key cryptography is widely used for the authentication protocols themselves and for establishing the session key.

The point of using a new, randomly-chosen session key for each new connection is to minimize the amount of traffic that gets sent with the users' secret keys or public keys, to reduce the amount of ciphertext an intruder can obtain, and to minimize the damage done if a process crashes and its core dump falls into the wrong hands. Hopefully, the only key present then will be the session key. All the permanent keys should have been carefully zeroed out after the session was established.

Authentication Based on a Shared Secret Key

For our first authentication protocol, we will assume that Alice and Bob already share a secret key, K_{AB} (In the formal protocols, we will abbreviate Alice as A and Bob as B, respectively.) This shared key might have been agreed upon on the telephone, or in person, but, in any event, not on the (insecure) network.

This protocol is based on a principle found in many authentication protocols: one party sends a random number to the other, who then transforms it in a special way and then returns the result. Such protocols are called **challenge-response** protocols. In this and subsequent authentication protocols, the following notation will be used:

A, B are the identities of Alice and Bob
R_i's are the challenges, where the subscript identifies the challenger
K_i are keys, where i indicates the owner; K_S is the session key

The message sequence for our first shared-key authentication protocol is shown in Fig. 7-12. In message 1, Alice sends her identity, A, to Bob in a way that Bob understands. Bob, of course, has no way of knowing whether this message came from Alice or from Trudy, so he chooses a challenge, a large random number, R_B, and sends it back to "Alice" as message 2, in plaintext. Alice then encrypts the message with the key she shares with Bob and sends the ciphertext, $K_{AB}(R_B)$, back in message 3. When Bob sees this message, he immediately knows that it came from Alice because Trudy does not know K_{AB} and thus could

---

[†] I thank Kaufman$_1$ et al.$_{23}$ (1995) for revealing her name.

not have generated it. Furthermore, since R_B was chosen randomly from a large space (say, 128-bit random numbers), it is very unlikely that Trudy would have seen R_B and its response from an earlier session.

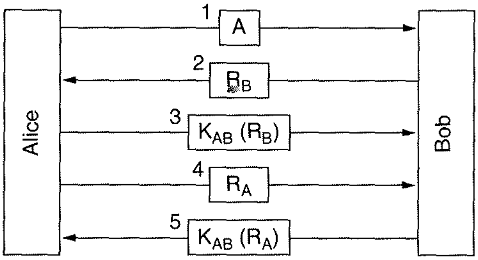


Fig. 7-12. Two-way authentication using a challenge-response protocol.

At this point, Bob is sure he is talking to Alice, but Alice is not sure of anything. For all Alice knows, Trudy might have intercepted message 1 and sent back R_B in response. Maybe Bob died last night. To find out whom she is talking to, Alice picks a random number, R_A and sends it to Bob as plaintext, in message 4. When Bob responds with $K_{AB}(R_A)$, Alice knows she is talking to Bob. If they wish to establish a session key now, Alice can pick one, K_S, and send it to Bob encrypted with K_{AB}.

Although the protocol of Fig. 7-12 works, it contains extra messages. These can be eliminated by combining information, as illustrated in Fig. 7-13. Here Alice initiates the challenge-response protocol instead of waiting for Bob to do it. Similarly, while he is responding to Alice's challenge, Bob sends his own. The entire protocol can be reduced to three messages instead of five.

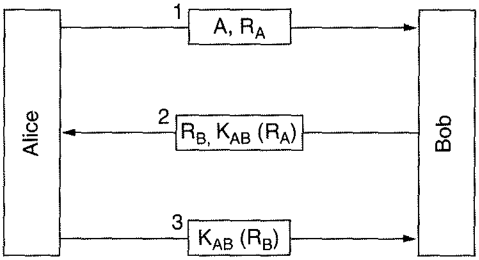


Fig. 7-13. A shortened two-way authentication protocol.

Is this new protocol an improvement over the original one? In one sense it is: it is shorter. Unfortunately, it is also wrong. Under certain circumstances, Trudy can defeat this protocol by using what is known as a **reflection attack**. In particular, Trudy can break it if it is possible to open multiple sessions with Bob at once. This situation would be true, for example, if Bob is a bank and is prepared to accept many simultaneous connections from teller machines at once.

Trudy's reflection attack is shown in Fig. 7-14. It starts out with Trudy claiming she is Alice and sending R_T. Bob responds, as usual, with his own challenge, R_B. Now Trudy is stuck. What can she do? She does not know $K_{AB}(R_B)$.

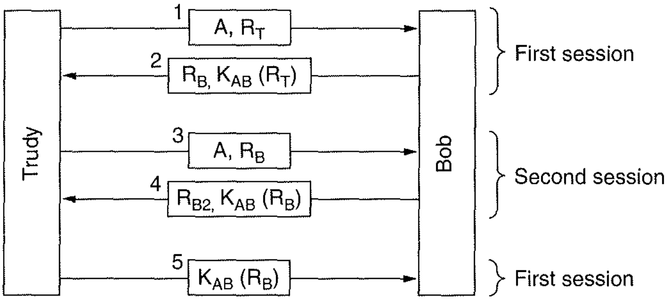


Fig. 7-14. The reflection attack.

She can open a second session with message 3, supplying the R_B taken from message 2 as her challenge. Bob calmly encrypts it and sends back $K_{AB}(R_B)$ in message 4. Now Trudy has the missing information, so she can complete the first session and abort the second one. Bob is now convinced that Trudy is Alice, so when she asks for her bank account balance, he gives it to her without question. Then when she asks him to transfer it all to a secret bank account in Switzerland, he does so without a moment's hesitation.

The moral of this story is:

Designing a correct authentication protocol is harder than it looks.

Three general rules that often help are as follows:

1. Have the initiator prove who she is before the responder has to. In this case, Bob gives away valuable information before Trudy has to give any evidence of who she is.
2. Have the initiator and responder use different keys for proof, even if this means having two shared keys, K_{AB} and K'_{AB}.
3. Have the initiator and responder draw their challenges from different sets. For example, the initiator must use even numbers and the responder must use odd numbers.

All three rules were violated here, with disastrous results. Note that our first (five-message) authentication protocol requires Alice to prove her identity first, so that protocol is not subject to the reflection attack.

Establishing a Shared Key: The Diffie-Hellman Key Exchange

So far we have assumed that Alice and Bob share a secret key. Suppose that they do not? How can they establish one? One way would be for Alice to call Bob and give him her key on the phone, but he would probably start out by saying: "How do I know you are Alice and not Trudy?" They could try to arrange a meeting, with each one bringing a passport, a drivers' license, and three major credit cards, but being busy people, they might not be able to find a mutually acceptable date for months. Fortunately, incredible as it may sound, there is a way for total strangers to establish a shared secret key in broad daylight, even with Trudy carefully recording every message.

The protocol that allows strangers to establish a shared secret key is called the **Diffie-Hellman key exchange** (Diffie and Hellman, 1976) and works as follows. Alice and Bob have to agree on two large prime numbers, n, and g, where $(n - 1)/2$ is also a prime and certain conditions apply to g. These numbers may be public, so either one of them can just pick n and g and tell the other openly. Now Alice picks a large (say, 512-bit) number, x, and keeps it secret. Similarly, Bob picks a large secret number, y.

Alice initiates the key exchange protocol by sending Bob a message containing $(n, g, g^x \bmod n)$, as shown in Fig. 7-15. Bob responds by sending Alice a message containing $g^y \bmod n$. Now Alice takes the number Bob sent her and raises it to the xth power to get $(g^y \bmod n)^x$. Bob performs a similar operation to get $(g^x \bmod n)^y$. By the laws of modular arithmetic, both calculations yield $g^{xy} \bmod n$. Lo and behold, Alice and Bob now share a secret key, $g^{xy} \bmod n$.

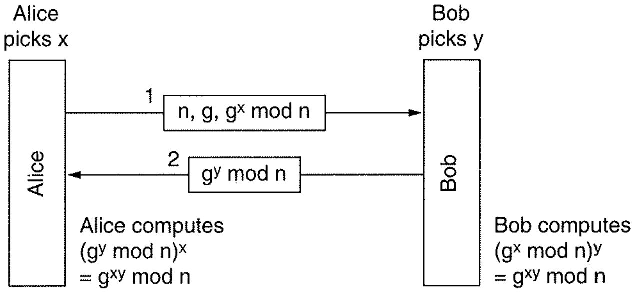


Fig. 7-15. The Diffie-Hellman key exchange.

Trudy, of course, has seen both messages. She knows g and n from message 1. If she could compute x and y, she could figure out the secret key. The trouble is, given only $g^x \bmod n$, she cannot find x. No practical algorithm for computing discrete logarithms modulo a very large prime number is known.

To make the above example more concrete, we will use the (completely unrealistic) values of $n = 47$ and $g = 3$. Alice picks $x = 8$ and Bob picks $y = 10$.

Both of these are kept secret. Alice's message to Bob is (47, 3, 28) because 3^8 mod 47 is 28. Bob's message to Alice is (17). Alice computes 17^8 mod 47, which is 4. Bob computes 28^{10} mod 47, which is 4. Alice and Bob have independently determined that the secret key is now 4. Trudy has to solve the equation $3^x \bmod 47 = 28$, which can be done by exhaustive search for small numbers like this, but not when all the numbers are hundreds of bits long. All currently-known algorithms simply take too long, even using a massively parallel supercomputer.

Despite the elegance of the Diffie-Hellman algorithm, there is a problem: when Bob gets the triple (47, 3, 28), how does he know it is from Alice and not from Trudy? There is no way he can know. Unfortunately, Trudy can exploit this fact to deceive both Alice and Bob, as illustrated in Fig. 7-16. Here, while Alice and Bob are choosing x and y, respectively, Trudy picks her own random number, z. Alice sends message 1 intended for Bob. Trudy intercepts it and sends message 2 to Bob, using the correct g and n (which are public anyway) but with her own z instead of x. She also sends message 3 back to Alice. Later Bob sends message 4 to Alice, which Trudy again intercepts and keeps.

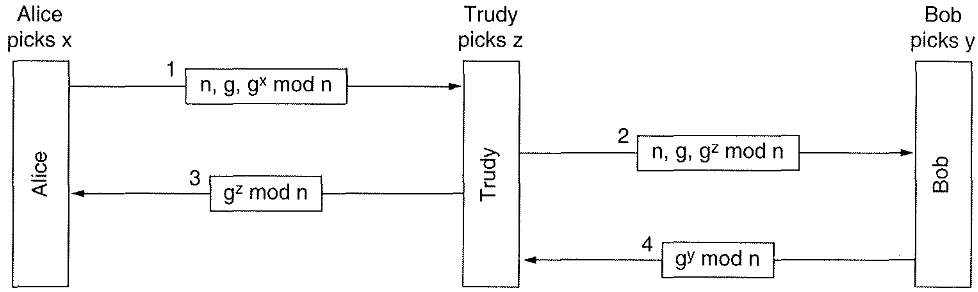


Fig. 7-16. The bucket brigade attack.

Now everybody does the modular arithmetic. Alice computes the secret key as g^{xz} mod n, and so does Trudy (for messages to Alice). Bob computes g^{yz} mod n and so does Trudy (for messages to Bob). Alice thinks she is talking to Bob so she establishes a session key (with Trudy). So does Bob. Every message that Alice sends on the encrypted session is captured by Trudy, stored, modified if desired, and then (optionally) passed on to Bob. Similarly in the other direction. Trudy sees everything and can modify all messages at will, while both Alice and Bob are under the illusion that they have a secure channel to one another. This attack is known as the **bucket brigade attack**, because it vaguely resembles an old-time volunteer fire department passing buckets along the line from the fire truck to the fire. It is also called the **(wo)man-in-the-middle attack**, which should not be confused with the meet-in-the-middle attack on block ciphers. Fortunately, more complex algorithms can defeat this attack.

Authentication Using a Key Distribution Center

Setting up a shared secret with a stranger almost worked, but not quite. On the other hand, it probably was not worth doing in the first place (sour grapes attack). To talk to *n* people this way, you would need *n* keys. For popular people, key management would become a real burden, especially if each key had to be stored on a separate plastic chip card.

A different approach is to introduce a trusted key distribution center (KDC). In this model, each user has a single key shared with the KDC. Authentication and session key management now goes through the KDC. The simplest known KDC authentication protocol involving two parties and a trusted KDC is depicted in Fig. 7-17.

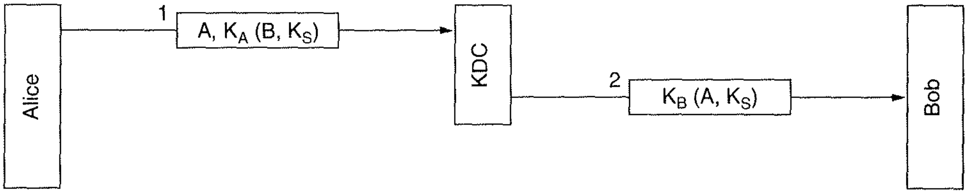


Fig. 7-17. A first attempt at an authentication protocol using a KDC.

The idea behind this protocol is simple: Alice picks a session key, K_S, and tells the KDC that she wants to talk to Bob using K_S. This message is encrypted with the secret key Alice shares (only) with the KDC, K_A. The KDC decrypts this message, extracting Bob's identity and the session key. It then constructs a new message containing Alice's identity and the session key and sends this message to Bob. This encryption is done with K_B, the secret key Bob shares with the KDC. When Bob decrypts the message, he learns that Alice wants to talk to him, and which key she wants to use.

The authentication here happens for free. The KDC knows that message 1 must have come from Alice, since no one else would have been able to encrypt it with Alice's secret key. Similarly, Bob knows that message 2 must have come from the KDC, whom he trusts, since no one else knows his secret key.

Unfortunately, this protocol has a serious flaw. Trudy needs some money, so she figures out some legitimate service she can perform for Alice, makes an attractive offer, and gets the job. After doing the work, Trudy then politely requests Alice to pay by bank transfer. Alice then establishes a session key with her banker, Bob. Then she sends Bob a message requesting money to be transferred to Trudy's account.

Meanwhile, Trudy is back to her old ways, snooping on the network. She copies both message 2 in Fig. 7-17, and the money-transfer request that follows it.

Later, she replays both of them to Bob. Bob gets them and thinks: "Alice must have hired Trudy again. She clearly does good work." Bob then transfers an equal amount of money from Alice's account to Trudy's. Some time after the 50th message pair, Bob runs out of the office to find Trudy to offer her a big loan so she can expand her obviously successful business. This problem is called the **replay attack**.

Several solutions to the replay attack are possible. The first one is to include a timestamp in each message. Then if anyone receives an obsolete message, it can be discarded. The trouble with this approach is that clocks are never exactly synchronized over a network, so there has to be some interval during which a timestamp is valid. Trudy can replay the message during this interval and get away with it.

The second solution is to put a one-time, unique message number, usually called a **nonce**, in each message. Each party then has to remember all previous nonces and reject any message containing a previously used nonce. But nonces have to be remembered forever, lest Trudy try replaying a 5-year-old message. Also, if some machine crashes and it loses its nonce list, it is again vulnerable to a replay attack. Timestamps and nonces can be combined to limit how long nonces have to be remembered, but clearly the protocol is going to get a lot more complicated.

A more sophisticated approach to authentication is to use a multiway challenge-response protocol. A well-known example of such a protocol is the **Needham-Schroeder authentication** protocol (Needham and Schroeder, 1978), one variant of which is shown in Fig. 7-18.

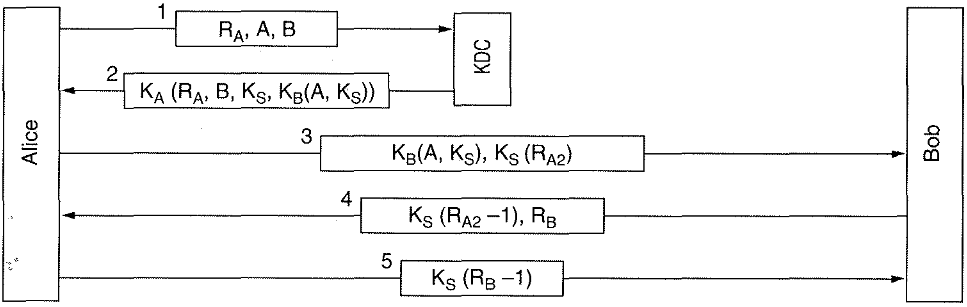


Fig. 7-18. The Needham-Schroeder authentication protocol.

The protocol begins with Alice telling the KDC that she wants to talk to Bob. This message contains a large random number, R_A, as a nonce. The KDC sends back message 2 containing Alice's random number, a session key, and a ticket that she can send to Bob. The point of the random number, R_A, is to assure Alice that message 2 is fresh, and not a replay. Bob's identity is also enclosed in case Trudy gets any funny ideas about replacing B in message 1 with her own identity so the KDC will encrypt the ticket at the end of message 2 with K_T instead of K_B. The ticket encrypted with K_B is included inside the encrypted message to prevent Trudy from replacing it with something else on the way back to Alice.

Alice now sends the ticket to Bob, along with a new random number, R_{A2}, encrypted with the session key, K_S. In message 4, Bob sends back $K_S(R_{A2} - 1)$ to prove to Alice that she is talking to the real Bob. Sending back $K_S(R_{A2})$ would not have worked, since Trudy could just have stolen it from message 3.

After receiving message 4, Alice is now convinced that she is talking to Bob, and that no replays could have been used so far. After all, she just generated R_{A2} a few milliseconds ago. The purpose of message 5 is to convince Bob that it is indeed Alice he is talking to, and no replays are being used here either. By having each party both generate a challenge and respond to one, the possibility of any kind of replay attack is eliminated.

Although this protocol seems pretty solid, it does have a slight weakness. If Trudy ever manages to obtain an old session key in plaintext, she can initiate a new session with Bob replaying the message 3 corresponding to the compromised key and convince him that she is Alice (Denning and Sacco, 1981). This time she can plunder Alice's bank account without having to perform the legitimate service even once.

Needham and Schroeder later published a protocol that corrects this problem (Needham and Schroeder, 1987). In the same issue of the same journal, Otway and Rees (1987) also published a protocol that solves the problem in a shorter way. Figure 7-19 shows a slightly modified Otway-Rees protocol.

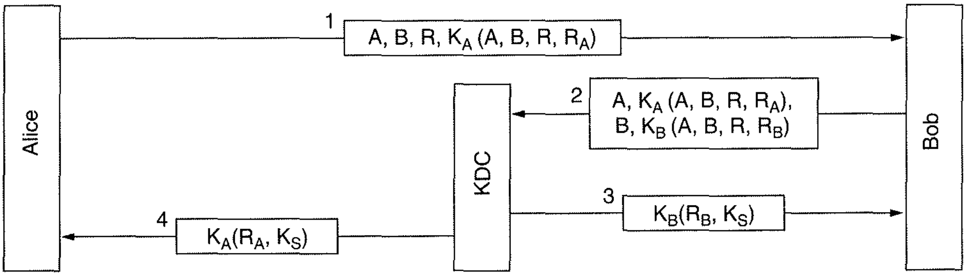


Fig. 7-19. The Otway-Rees authentication protocol (slightly simplified).

In the Otway-Rees protocol, Alice starts out by generating a pair of random numbers, R, which will be used as a common identifier, and R_A which Alice will use to challenge Bob. When Bob gets this message, he constructs a new message from the encrypted part of Alice's message, and an analogous one of his own. Both the parts encrypted with K_A and K_B identify Alice and Bob, contain the common identifier, and contain a challenge.

The KDC checks to see if the R in both parts is the same. It might not be because Trudy tampered with R in message 1 or replaced part of message 2. If the two *R*s match, the KDC believes that the request message from Bob is valid. It then generates a session key and encrypts it twice, once for Alice and once for Bob. Each message contains the receiver's random number, as proof that the KDC, and not Trudy, generated the message. At this point both Alice and Bob are in possession of the same session key and can start communicating. The first time they exchange data messages, each one can see that the other one has an identical copy of K_S, so the authentication is then complete.

Authentication Using Kerberos

An authentication protocol used in many real systems is **Kerberos**, which is based on a variant of Needham-Schroeder. It is named for a multiheaded dog in Greek Mythology that used to guard the entrance to Hades (presumably to keep undesirables out). Kerberos was designed at M.I.T. to allow workstation users to access network resources in a secure way. Its biggest difference with Needham-Schroeder is its assumption that all clocks are fairly-well synchronized. The protocol has gone through several iterations. V4 is the version most widely used in industry, so we will describe it. Afterward, we will say a few words about its successor, V5. For more information, see (Neuman and Ts'o, 1994; and Steiner et al., 1988).

Kerberos involves three servers in addition to Alice (a client workstation):

Authentication Server (AS): verifies users during login
Ticket-Granting Server (TGS): issues "proof of identity tickets"
Bob the server: actually does the work Alice wants performed

AS is similar to a KDC in that it shares a secret password with every user. The TGS's job is to issue tickets that can convince the real servers that the bearer of a TGS ticket really is who he or she claims to be.

To start a session, Alice sits down at a arbitrary public workstation and types her name. The workstation sends her name to the AS in plaintext, as shown in Fig. 7-20. What comes back is a session key and a ticket, $K_{TGS}(A, K_S)$, intended for the TGS. These items are packaged together and encrypted using Alice's secret key, so that only Alice can decrypt them. Only when message 2 arrives, does the workstation ask for Alice's password. The password is then used to generate K_A, in order to decrypt message 2 and obtain the session key and TGS ticket inside it. At this point, the workstation overwrites Alice's password, to make sure that it is only inside the workstation for a few milliseconds at most. If Trudy tries logging in as Alice, the password she types will be wrong and the workstation will detect this because the standard part of message 2 will be incorrect.

After she logs in, Alice may tell the workstation that she wants to contact Bob the file server. The workstation then sends message 3 to the TGS asking for a ticket to use with Bob. The key element in this request is $K_{TGS}(A, K_S)$, which is

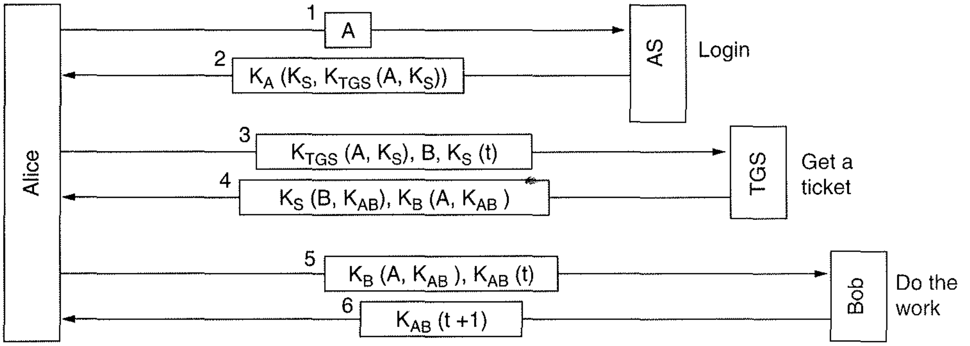


Fig. 7-20. The operation of Kerberos V4.

encrypted with the TGS's secret key and is used as proof that the sender really is Alice. The TGS responds by creating a session key, K_{AB}, for Alice to use with Bob. Two versions of it are sent back. The first is encrypted with only K_S, so Alice can read it. The second is encrypted with Bob's key, K_B, so Bob can read it.

Trudy can copy message 3 and try to use it again, but she will be foiled by the encrypted timestamp, *t*, sent along with it. Trudy cannot replace the timestamp with a more recent one, because she does not know K_S, the session key Alice uses to talk to the TGS. Even if Trudy replays message 3 quickly, all she will get is another copy of message 4, which she could not decrypt the first time and will not be able to decrypt the second time either.

Now Alice can send K_{AB} to Bob to establish a session with him. This exchange is also timestamped. The response is proof to Alice that she is actually talking to Bob, not to Trudy.

After this series of exchanges, Alice can communicate with Bob under cover of K_{AB}. If she later decides she needs to talk to another server, Carol, she just repeats message 3 to the TGS, only now specifying *C* instead of *B*. The TGS will promptly respond with a ticket encrypted with K_C that Alice can send to Carol and that Carol will accept as proof that it came from Alice.

The point of all this work is that now Alice can access servers all over the network in a secure way, and her password never has to go over the network. In fact, it only had to be in her own workstation for a few milliseconds. However, note that each server does its own authorization. When Alice presents her ticket to Bob, this merely proves to Bob who sent it. Precisely what Alice is allowed to do is up to Bob.

Since the Kerberos designers did not expect the entire world to trust a single authentication server, they made provision for having multiple **realms**, each with its own AS and TGS. To get a ticket for a server in a distant realm, Alice would ask her own TGS for a ticket accepted by the TGS in the distant realm. If the distant TGS has registered with the local TGS (the same way local servers do), the local TGS will give Alice a ticket valid at the distant TGS. She can then do business over there, such as getting tickets for servers in that realm. Note, however, that for parties in two realms to do business, each one must trust the other's TGS.

Kerberos V5 is fancier than V4 and has more overhead. It also uses OSI ASN.1 (Abstract Syntax Notation 1) for describing data types and has small changes in the protocols. Furthermore, it has longer ticket lifetimes, allows tickets to be renewed, and will issue postdated tickets. In addition, at least in theory, it is not DES dependent, as V4 is, and supports multiple realms.

Authentication Using Public-Key Cryptography

Mutual authentication can also be done using public-key cryptography. To start with, let us assume Alice and Bob already know each other's public keys (a nontrivial issue). They want to establish a session, and then use secret-key cryptography on that session, since it is typically 100 to 1000 times faster than public-key cryptography. The purpose of the initial exchange then is to authenticate each other and agree on a secret shared session key.

This setup can be done is various ways. A typical one is shown in Fig. 7-21. Here Alice starts by encrypting her identity and a random number, R_A, using Bob's public (or encryption) key, E_B. When Bob receives this message, he has no idea of whether it came from Alice or from Trudy, but he plays along and sends Alice back a message containing Alice's R_A, his own random number, R_B, and a proposed session key, K_S.

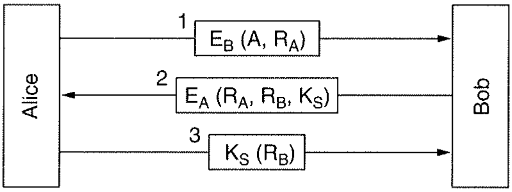


Fig. 7-21. Mutual authentication using public-key cryptography.

When Alice gets message 2, she decrypts it using her private key. She sees R_A in it, which gives her a warm feeling inside. The message must have come from Bob, since Trudy has no way of determining R_A. Furthermore, it must be fresh and not a replay, since she just sent Bob R_A. Alice agrees to the session by sending back message 3. When Bob sees R_B encrypted with the session key he just generated, he knows Alice got message 2 and verified R_A.

What can Trudy do to try to subvert this protocol? She can fabricate message 1 and trick Bob into probing Alice, but Alice will see an R_A that she did not send and will not proceed further. Trudy cannot forge message 3 convincingly because she does not know R_B or K_S and cannot determine them without Alice's private key. She is out of luck.

However, the protocol does have a weakness: it assumes that Alice and Bob already know each other's public keys. Suppose that they do not. Alice could just send Bob her public key in the first message and ask Bob to send his back in the next one. The trouble with this approach is that it is subject to a bucket brigade attack. Trudy can capture Alice's message to Bob and send her own public key back to Alice. Alice will think she has a key for talking to Bob, when, in fact, she has a key for talking to Trudy. Now Trudy can read all the messages encrypted with what Alice thinks is Bob's public key.

The initial public-key exchange can be avoided by having all the public keys stored in a public database. Then Alice and Bob can fetch each other's public keys from the database. Unfortunately, Trudy can still pull off the bucket brigade attack by intercepting the requests to the database and sending simulated replies containing her own public key. After all, how do Alice and Bob know that the replies came from the real data base and not from Trudy?

Rivest and Shamir (1984) have devised a protocol that foils Trudy's bucket brigade attack. In their **interlock protocol**, after the public key exchange, Alice sends only half of her message to Bob, say, only the even bits (after encryption). Bob then responds with his even bits. After getting Bob's even bits, Alice sends her odd bits, then Bob does too.

The trick here is that when Trudy gets Alice's even bits, she cannot decrypt the message, even though Trudy has the private key. Consequently, she is unable to reencrypt the even bits using Bob's public key. If she sends junk to Bob, the protocol will continue, but Bob will shortly discover that the fully assembled message makes no sense and realized that he has been spoofed.

7.1.6. Digital Signatures

The authenticity of many legal, financial, and other documents is determined by the presence or absence of an authorized handwritten signature. And photocopies do not count. For computerized message systems to replace the physical transport of paper and ink documents, a solution must be found to these problems.

The problem of devising a replacement for handwritten signatures is a difficult one. Basically, what is needed is a system by which one party can send a "signed" message to another party in such a way that

1. The receiver can verify the claimed identity of the sender.
2. The sender cannot later repudiate the contents of the message.
3. The receiver cannot possibly have concocted the message himself.

The first requirement is needed, for example, in financial systems. When a customer's computer orders a bank's computer to buy a ton of gold, the bank's computer needs to be able to make sure that the computer giving the order really belongs to the company whose account is to be debited.

The second requirement is needed to protect the bank against fraud. Suppose that the bank buys the ton of gold, and immediately thereafter the price of gold drops sharply. A dishonest customer might sue the bank, claiming that he never issued any order to buy gold. When the bank produces the message in court, the customer denies having sent it.

The third requirement is needed to protect the customer in the event that the price of gold shoots up and the bank tries to construct a signed message in which the customer asked for one bar of gold instead of one ton.

Secret-Key Signatures

One approach to digital signatures is to have a central authority that knows everything and whom everyone trusts, say Big Brother (*BB*). Each user then chooses a secret key and carries it by hand to *BB*'s office. Thus only Alice and *BB* know Alice's secret, K_A, and so on.

When Alice wants to send a signed plaintext message, *P*, to her banker, Bob, she generates $K_A(B, R_A, t, P)$ and sends it as depicted in Fig. 7-22. *BB* sees that the message is from Alice, decrypts it, and sends a message to Bob as shown. The message to Bob contains the plaintext of Alice's message and also the signed message $K_{BB}(A, t, P)$, where *t* is a timestamp. Bob now carries out Alice's request.

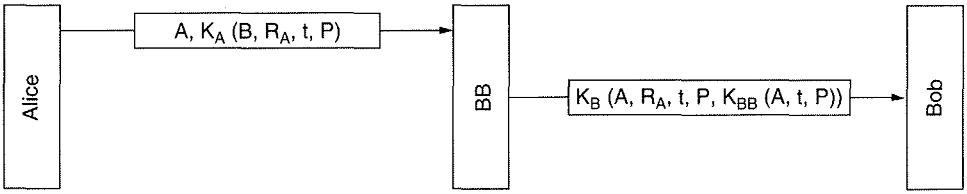


Fig. 7-22. Digital signatures with Big Brother.

What happens if Alice later denies sending the message? Step 1 is that everyone sues everyone (at least, in the United States). Finally, when the case comes to court and Alice vigorously denies sending Bob the disputed message, the judge will ask Bob how he can be sure that the disputed message came from Alice and not from Trudy. Bob first points out that *BB* will not accept a message from Alice unless it is encrypted with K_A, so there is no possibility of Trudy sending *BB* a false message from Alice.

Bob then dramatically produces Exhibit A, $K_{BB}(A, t, P)$. Bob says that this is a message signed by *BB* which proves Alice sent *P* to Bob. The judge then asks

BB (whom everyone trusts) to decrypt Exhibit A. When *BB* testifies that Bob is telling the truth, the judge decides in favor of Bob. Case dismissed.

One potential problem with the signature protocol of Fig. 7-22 is Trudy replaying either message. To minimize this problem, timestamps are used throughout. Furthermore, Bob can check all recent messages to see if R_A was used in any of them. If so, the message is discarded as a replay. Note that Bob will reject very old messages based on the timestamp. To guard against instant replay attacks, Bob just checks the R_A of every incoming message to see if such a message has been received from Alice in the past hour. If not, Bob can safely assume this is a new request.

Public-Key Signatures

A structural problem with using secret-key cryptography for digital signatures is that everyone has to agree to trust Big Brother. Furthermore, Big Brother gets to read all signed messages. The most logical candidates for running the Big Brother server are the government, the banks, or the lawyers. These organizations do not inspire total confidence in all citizens. Hence, it would be nice if signing documents did not require a trusted authority.

Fortunately, public-key cryptography can make an important contribution here. Let us assume that the public-key encryption and decryption algorithms have the property that $E(D(P)) = P$ in addition to the usual property that $D(E(P)) = P$. (RSA has this property, so the assumption is not unreasonable.) Assuming that this is the case, Alice can send a signed plaintext message, P, to Bob by transmitting $E_B(D_A(P))$. Note carefully that Alice knows her own (private) decryption key, D_A, as well as Bob's public key, E_B, so constructing this message is something Alice can do.

When Bob receives the message, he transforms it using his private key, as usual, yielding $D_A(P)$, as shown in Fig. 7-23. He stores this text in a safe place and then decrypts it using E_A to get the original plaintext.

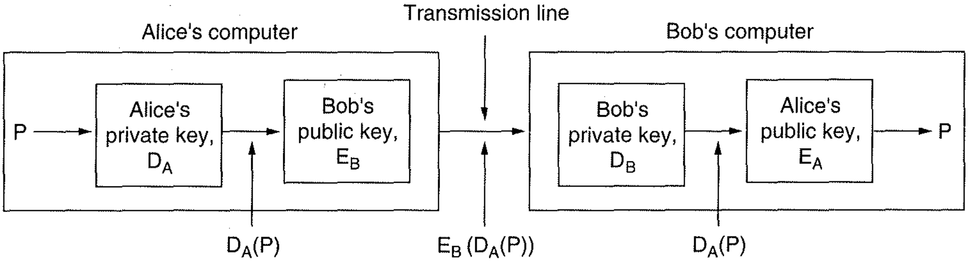


Fig. 7-23. Digital signatures using public-key cryptography.

To see how the signature property works, suppose that Alice subsequently denies having sent the message P to Bob. When the case comes up in court, Bob can produce both P and $D_A(P)$. The judge can easily verify that Bob indeed has a valid message encrypted by D_A by simply applying E_A to it. Since Bob does not know what Alice's private key is, the only way Bob could have acquired a message encrypted by it is if Alice did indeed send it. While in jail for perjury and fraud, Alice will have plenty of time to devise interesting new public-key algorithms.

Although using public-key cryptography for digital signatures is an elegant scheme, there are problems that are related to the environment in which they operate rather than with the basic algorithm. For one thing, Bob can prove that a message was sent by Alice only as long as D_A remains secret. If Alice discloses her secret key, the argument no longer holds, because anyone could have sent the message, including Bob himself.

The problem might arise, for example, if Bob is Alice's stockbroker. Alice tells Bob to buy a certain stock or bond. Immediately thereafter, the price drops sharply. To repudiate her message to Bob, Alice runs to the police claiming that her home was burglarized and her key was stolen. Depending on the laws in her state or country, she may or may not be legally liable, especially if she claims not to have discovered the break-in until getting home from work, several hours later.

Another problem with the signature scheme is what happens if Alice decides to change her key. Doing so is clearly legal, and it is probably a good idea to do so periodically. If a court case later arises, as described above, the judge will apply the *current* E_A to $D_A(P)$ and discover that it does not produce P. Bob will look pretty stupid at this point. Consequently, it appears that some authority is probably needed to record all key changes and their dates.

In principle, any public-key algorithm can be used for digital signatures. The de facto industry standard is the RSA algorithm. Many security products use it. However, in 1991, NIST (National Institute of Standards and Technology) proposed using a variant of the El Gamal public-key algorithm for their new **Digital Signature Standard (DSS)**. El Gamal gets its security from the difficulty of computing discrete logarithms, rather than the difficulty of factoring large numbers.

As usual when the government tries to dictate cryptographic standards, there was an uproar. DSS was criticized for being

1. Too secret (NSA designed the protocol for using El Gamal).
2. Too new (El Gamal has not yet been thoroughly analyzed).
3. Too slow (10 to 40 times slower than RSA for checking signatures).
4. Too insecure (fixed 512-bit key).

In a subsequent revision, the fourth point was rendered moot when keys up to 1024 bits were allowed. It is not yet clear whether DSS will catch on. For more details, see (Kaufman et al., 1995; Schneier, 1996; and Stinson, 1995).

Message Digests

One criticism of signature methods is that they often couple two distinct functions: authentication and secrecy. Often, authentication is needed but secrecy is not. Since cryptography is slow, it is frequently desirable to be able to send signed plaintext documents. Below we will describe an authentication scheme that does not require encrypting the entire message (De Jonge and Chaum, 1987).

This scheme is based on the idea of a one-way hash function that takes an arbitrarily long piece of plaintext and from it computes a fixed-length bit string. This hash function, often called a **message digest**, has three important properties:

1. Given P, it is easy to compute $MD(P)$.
2. Given $MD(P)$, it is effectively impossible to find P.
3. No one can generate two messages that have the same message digest.

To meet criterion 3, the hash should be at least 128 bits long, preferably more.

Computing a message digest from a piece of plaintext is much faster than encrypting that plaintext with a public-key algorithm, so message digests can be used to speed up digital signature algorithms. To see how this works, consider the signature protocol of Fig. 7-22 again. Instead of signing P with $K_{BB}(A, t, P)$, BB now computes the message digest by applying MD to P, yielding $MD(P)$. BB then encloses $K_{BB}(A, t, MD(P))$ as the fifth item in the list encrypted with K_B that is sent to Bob, instead of $K_{BB}(A, t, P)$.

If a dispute arises, Bob can produce both P and $K_{BB}(A, t, MD(P))$. After Big Brother has decrypted it for the judge, Bob has $MD(P)$, which is guaranteed to be genuine, and the alleged P. However, since it is effectively impossible for Bob to find any other message that gives this hash, the judge will easily be convinced that Bob is telling the truth. Using message digests in this way saves both encryption time and message transport and storage costs.

Message digests work in public-key cryptosystems, too, as shown in Fig. 7-24. Here, Alice first computes the message digest of her plaintext. She then signs the message digest and sends both the signed digest and the plaintext to Bob. If Trudy replaces P underway, Bob will see this when he computes $MD(P)$ himself.

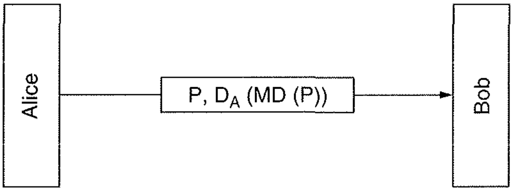


Fig. 7-24. Digital signatures using message digests.

A variety of message digest functions have been proposed. The most widely used ones are MD5 (Rivest, 1992) and SHA (NIST, 1993). **MD5** is the fifth in a series of hash functions designed by Ron Rivest. It operates by mangling bits in a sufficiently complicated way that every output bit is affected by every input bit. Very briefly, it starts out by padding the message to a length of 448 bits (modulo 512). Then the original length of the message is appended as a 64-bit integer to give a total input whose length is a multiple of 512 bits. The last precomputation step is initializing a 128-bit buffer to a fixed value.

Now the computation starts. Each round takes a 512-bit block of input and mixes it thoroughly with the 128-bit buffer. For good measure, a table constructed from the sine function is also thrown in. The point of using a known function like the sine is not because it is more random than a random number generator, but to avoid any suspicion that the designer built in a clever trapdoor through which only he can enter. IBM's refusal to disclose the principles behind the design of the S-boxes in DES led to a great deal of speculation about trapdoors. Four rounds are performed per input block. This process continues until all the input blocks have been consumed. The contents of the 128-bit buffer form the message digest. The algorithm has been optimized for software implementation on 32-bit machines. As a consequence, it may not be fast enough for future high-speed networks (Touch, 1995).

The other major message digest function is **SHA** (**Secure Hash Algorithm**), developed by NSA and blessed by NIST. Like MD5, it processes input data in 512-bit blocks, only unlike MD5, it generates a 160-bit message digest. It starts out by padding the message, then adding a 64-bit length to get a multiple of 512 bits. Then it initializes its 160-bit output buffer.

For each input block, the output buffer is updated using the 512-bit input block. No table of random numbers (or sine function values) is used, but for each block 80 rounds are computed, resulting in a thorough mixing. Each group of 20 rounds uses different mixing functions.

Since SHA's hash code is 32 bits longer than MD5's, all other things being equal, it is a factor of 2^{32} more secure than MD5. However, it is also slower than MD5, and having a hash code that is not a power of two might sometimes be an inconvenience. Otherwise, the two are roughly similar technically. Politically, MD5 is defined in an RFC and used heavily on the Internet. SHA is a government standard, and used by companies that have to use it because the government tells them to, or by those that want the extra security. A revised version, SHA-1, has been approved as a standard by NIST.

The Birthday Attack

In the world of crypto, nothing is ever what it seems to be. One might think that it would take on the order of 2^m operations to subvert an m-bit message digest. In fact, $2^{m/2}$ operations will often do using the **birthday attack,** an approach published by Yuval (1979) in his now-classic paper "How to Swindle Rabin."

The idea for this attack comes from a technique that math professors often use in their probability courses. The question is: How many students do you need in a class before the probability of having two people with the same birthday exceeds 1/2? Most students expect the answer to be way over 100. In fact, probability theory says it is just 23. Without giving a rigorous analysis, intuitively, with 23 people, we can form $(23 \times 22)/2 = 253$ different pairs, each of which has a probability of 1/365 of being a hit. In this light, it is not really so surprising any more.

More generally, if there is some mapping between inputs and outputs with n inputs (people, messages, etc.) and k possible outputs (birthdays, message digests, etc.), there are $n(n-1)/2$ input pairs. If $n(n-1)/2 > k$, the chance of having at least one match is pretty good. Thus, approximately, a match is likely for $n > \sqrt{k}$. This result means that a 64-bit message digest can probably be broken by generating about 2^{32} messages and looking for two with the same message digest.

Let us look at a practical example. The Dept. of Computer Science at State University has one position for a tenured faculty member and two candidates, Tom and Dick. Tom was hired two years before Dick, so he goes up for review first. If he gets it, Dick is out of luck. Tom knows that the department chairperson, Marilyn, thinks highly of his work, so he asks her to write him a letter of recommendation to the Dean, who will decide on Tom's case. Once sent, all letters become confidential.

Marilyn tells her secretary, Ellen, to write the Dean a letter, outlining what she wants in it. When it is ready, Marilyn will review it, compute and sign the 64-bit digest, and send it to the Dean. Ellen can send the letter later by email.

Unfortunately for Tom, Ellen is romantically involved with Dick and would like to do Tom in, so she writes the letter below with the 32 bracketed options.

Dear Dean Smith,

This [*letter* | *message*] is to give my [*honest* | *frank*] opinion of Prof. Tom Wilson, who is [*a candidate* | *up*] for tenure [*now* | *this year*]. I have [*known* | *worked with*] Prof. Wilson for [*about* | *almost*] six years. He is an [*outstanding* | *excellent*] researcher of great [*talent* | *ability*] known [*worldwide* | *internationally*] for his [*brilliant* | *creative*] insights into [*many* | *a wide variety of*] [*difficult* | *challenging*] problems.

He is also a [*highly* | *greatly*] [*respected* | *admired*] [*teacher* | *educator*]. His students give his [*classes* | *courses*] [*rave* | *spectacular*] reviews. He is [*our* | *the Department's*] [*most popular* | *best-loved*] [*teacher* | *instructor*].

[*In addition* | *Additionally*] Prof. Wilson is a [*gifted* | *effective*] fund raiser. His [*grants* | *contracts*] have brought a [*large* | *substantial*] amount of money into [*the* | *our*] Department. [*This money has* | *These funds have*] [*enabled* | *permitted*] us to [*pursue* | *carry out*] many [*special* | *important*] programs, [*such as* | *for example*] your State 2000 program. Without these funds we would [*be unable* | *not be able*] to continue this program, which is so [*important* | *essential*] to both of us. I strongly urge you to grant him tenure.

Unfortunately for Tom, as soon as Ellen finishes composing and typing in this letter, she also writes a second one:

Dear Dean Smith,

This [*letter* | *message*] is to give my [*honest* | *frank*] opinion of Prof. Tom Wilson, who is [*a candidate* | *up*] for tenure [*now* | *this year*]. I have [*known* | *worked with*] Tom for [*about* | *almost*] six years. He is a [*poor* | *weak*] researcher not well known in his [*field* | *area*]. His research [*hardly ever* | *rarely*] shows [*insight in* | *understanding of*] the [*key* | *major*] problems of [*the* | *our*] day.

Furthermore, he is not a [*respected* | *admired*] [*teacher* | *educator*]. His students give his [*classes* | *courses*] [*poor* | *bad*] reviews. He is [*our* | *the Department's*] least popular [*teacher* | *instructor*], known [*mostly* | *primarily*] within [*the* | *our*] Department for his [*tendency* | *propensity*] to [*ridicule* | *embarrass*] students [*foolish* | *imprudent*] enough to ask questions in his classes.

[*In addition* | *Additionally*] Tom is a [*poor* | *marginal*] fund raiser. His [*grants* | *contracts*] have brought only a [*meager* | *insignificant*] amount of money into [*the* | *our*] Department. Unless new [*money is* | *funds are*] quickly located, we may have to cancel some essential programs, such as your State 2000 program. Unfortunately, under these [*conditions* | *circumstances*] I cannot in good [*conscience* | *faith*] recommend him to you for [*tenure* | *a permanent position*].

Now Ellen sets up her computer to compute the 2^{32} message digests of each letter overnight. Chances are, one digest of the first letter will match one digest of the second letter. If not, she can add a few more options and try again during the weekend. Suppose that she finds a match. Call the "good" letter A and the "bad" one B.

Ellen now emails letter A to Marilyn for her approval. Marilyn, of course, approves, computes her 64-bit message digest, signs the digest, and emails the signed digest off to Dean Smith. Independently, Ellen emails letter B to the Dean.

After getting the letter and signed message digest, the Dean runs the message digest algorithm on letter B, sees that it agrees with what Marilyn sent him, and fires Tom. (Optional ending: Ellen tells Dick what she did. Dick is appalled and breaks off with her. Ellen is furious and confesses to Marilyn. Marilyn calls the Dean. Tom gets tenure after all.) With MD5 the birthday attack is infeasible because even at 1 billion digests per second, it would take over 500 years to compute all 2^{64} digests of two letters with 64 variants each, and even then a match is not guaranteed.

7.1.7. Social Issues

The implications of network security for individual privacy and society in general are staggering. Below we will just mention a few of the salient issues.

Governments do not like citizens keeping secrets from them. In some countries (e.g., France) all nongovernmental cryptography is simply forbidden unless the government is given all the keys being used. As Kahn (1980) and Selfridge and Schwartz (1980) point out, government eavesdropping has been practiced on a far more massive scale than most people could dream of, and governments want more than just a pile of indecipherable bits for their efforts.

The U.S. government has proposed an encryption scheme for future digital telephones that includes a special feature to allow the police to tap and decrypt all telephone calls made in the United States. The government promises not to use this feature without a court order, but many people still remember how former FBI Director J. Edgar Hoover illegally tapped the telephones of Martin Luther King, Jr. and other people in an attempt to neutralize them. The police say they need this power to catch criminals. The debate on both sides is vehement, to put it mildly. A discussion of the technology involved (Clipper) is given in (Kaufman et al., 1995). A way to circumvent this technology and send messages that the government cannot read is described in (Blaze, 1994; and Schneier, 1996). Position statements on all sides are given in (Hoffman, 1995).

The United States has a law (22 U.S.C. 2778) that prohibits citizens from exporting munitions (war materiel), such as tanks and jet fighters, without authorization from the DoD. For purposes of this law, cryptographic software is classified as a munition. Phil Zimmermann, who wrote PGP (Pretty Good Privacy), an email protection program, has been accused of violating this law, even though the government admits that he did not export it (but he did give it to a friend who put it on the Internet where foreigners could obtain it). Many people regarded this widely-publicized incident as a gross violation of the rights of an American citizen working to enhance people's privacy.

Not being an American does not help. On July 9, 1986, three Israeli researchers working at the Weizmann Institute in Israel filed a U.S. patent application for a new digital signature scheme that they had invented. They spent the next 6 months discussing their research at conferences all over the world. On Jan. 6, 1987, the U.S. patent office told them to notify all Americans who knew about their results that disclosure of the research would subject them to two years in prison, a 10,000-dollar fine, or both. The patent office also wanted a list of all foreign nationals who knew about the research. To find out how this story turned out, see (Landau, 1988).

Patents are another hot topic. Nearly all public-key algorithms are patented. Patent protection lasts for 17 years. The RSA patent, for example, expires on Sept. 20, 2000.

Network security is politicized to an extent few other technical issues are, and rightly so, since it relates to the difference between a democracy and a police state in the digital era. The March 1993 and November 1994 issues of *Communications of the ACM* have long sections on telephone and network security, respectively, with vigorous arguments explaining and defending many points of view. Chapter 25 of Schneier's security book deals with the politics of cryptography

(Schneier, 1996). Chapter 8 of his email book does too (Schneier, 1995). Privacy and computers are also discussed in (Adam, 1995). These references are highly recommended for readers who wish to pursue their study of this subject.

7.2. DNS—Domain Name System

Programs rarely refer to hosts, mailboxes, and other resources by their binary network addresses. Instead of binary numbers, they use ASCII strings, such as *tana@art.ucsb.edu.* Nevertheless, the network itself only understands binary addresses, so some mechanism is required to convert the ASCII strings to network addresses. In the following sections we will study how this mapping is accomplished in the Internet.

Way back in the ARPANET, there was simply a file, *hosts.txt*, that listed all the hosts and their IP addresses. Every night, all the hosts would fetch it from the site at which it was maintained. For a network of a few hundred large timesharing machines, this approach worked reasonably well.

However, when thousands of workstations were connected to the net, everyone realized that this approach could not continue to work forever. For one thing, the size of the file would become too large. However, even more important, host name conflicts would occur constantly unless names were centrally managed, something unthinkable in a huge international network. To solve these problems, **DNS** (the **Domain Name System**) was invented.

The essence of DNS is the invention of a hierarchical, domain-based naming scheme and a distributed database system for implementing this naming scheme. It is primarily used for mapping host names and email destinations to IP addresses but can also be used for other purposes. DNS is defined in RFCs 1034 and 1035.

Very briefly, the way DNS is used is as follows. To map a name onto an IP address, an application program calls a library procedure called the **resolver**, passing it the name as a parameter. The resolver sends a UDP packet to a local DNS server, which then looks up the name and returns the IP address to the resolver, which then returns it to the caller. Armed with the IP address, the program can then establish a TCP connection with the destination, or send it UDP packets.

7.2.1. The DNS Name Space

Managing a large and constantly changing set of names is a nontrivial problem. In the postal system, name management is done by requiring letters to specify (implicitly or explicitly) the country, state or province, city, and street address of the addressee. By using this kind of hierarchical addressing, there is no confusion between the Marvin Anderson on Main St. in White Plains, N.Y. and the Marvin Anderson on Main St. in Austin, Texas. DNS works the same way.

7.6. THE WORLD WIDE WEB

The World Wide Web is an architectural framework for accessing linked documents spread out over thousands of machines all over the Internet. In 5 years, it went from being a way to distribute high-energy physics data to the application that millions of people think of as being "The Internet." Its enormous popularity stems from the fact that it has a colorful graphical interface that is easy for beginners to use, and it provides an enormous wealth of information on almost every conceivable subject, from aboriginals to zoology.

The Web (also known as **WWW**) began in 1989 at CERN, the European center for nuclear research. CERN has several accelerators at which large teams of scientists from the participating European countries carry out research in particle physics. These teams often have members from half a dozen or more countries. Most experiments are highly complex, and require years of advance planning and equipment construction. The Web grew out of the need to have these large teams of internationally dispersed researchers collaborate using a constantly changing collection of reports, blueprints, drawings, photos, and other documents.

The initial proposal for a web of linked documents came from CERN physicist Tim Berners-Lee in March 1989. The first (text-based) prototype was operational 18 months later. In December 1991, a public demonstration was given at the Hypertext '91 conference in San Antonio, Texas. Development continued during the next year, culminating in the release of the first graphical interface, Mosaic, in February 1993 (Vetter et al., 1994).

Mosaic was so popular that a year later, its author, Marc Andreessen left the National Center for Supercomputing Applications, where Mosaic was developed, to form a company, Netscape Communications Corp., whose goal was to develop clients, servers, and other Web software. When Netscape went public in 1995, investors, apparently thinking this was the next Microsoft, paid 1.5 billion dollars for the stock. This record was all the more surprising because the company had only one product, was operating deeply in the red, and had announced in its prospectus that it did not expect to make a profit for the foreseeable future.

In 1994, CERN and M.I.T. signed an agreement setting up the World Wide Web Consortium, an organization devoted to further developing the Web, standardizing protocols, and encouraging interoperability between sites. Berners-Lee became the director. Since then, hundreds of universities and companies have joined the consortium. M.I.T. runs the U.S. part of the consortium and the French research center, INRIA, runs the European part. Although there are more books about the Web than you can shake a stick at, the best place to get up-to-date information about the Web is (naturally) on the Web itself. The consortium's home page can be found at *http://www.w3.org* . Interested readers are referred there for links to pages covering all of the consortium's documents and activities.

In the following sections we will describe how the Web appears to the user, and, especially, how it works inside. Since the Web is basically a client-server system, we will discuss both the client (i.e., user) side and the server side. Then we will examine the language in which Web pages are written (HTML and Java). Finally, comes an examination of how to find information on the Web.

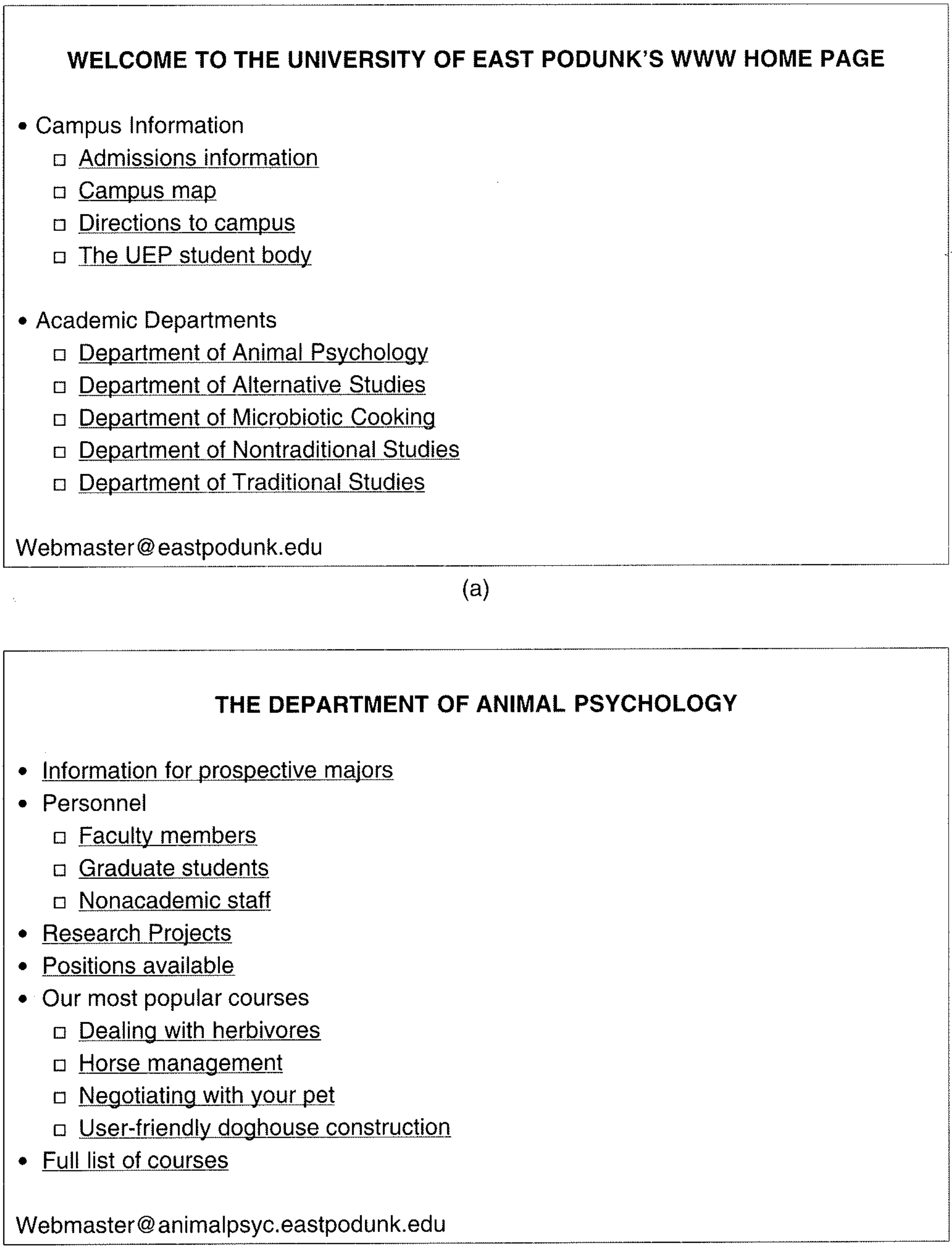
7.6.1. The Client Side

From the users' point of view, the Web consists of a vast, worldwide collection of documents, usually just called **pages** for short. Each page may contain links (pointers) to other, related pages, anywhere in the world. Users can follow a link (e.g., by clicking on it), which then takes them to the page pointed to. This process can be repeated indefinitely, possibly traversing hundreds of linked pages while doing so. Pages that point to other pages are said to use **hypertext.**

Pages are viewed with a program called a **browser**, of which Mosaic and Netscape are two popular ones. The browser fetches the page requested, interprets the text and formatting commands that it contains, and displays the page, properly formatted, on the screen. An example is given in Fig. 7-58(a). Like many Web pages, this one starts with a title, contains some information, and ends with the email address of the page's maintainer. Strings of text that are links to other pages, called **hyperlinks**, are highlighted, either by underlining, displaying them in a special color, or both. To follow a link, the user places the cursor on the highlighted area (using the mouse or the arrow keys) and selects it (by clicking a mouse button or hitting ENTER). Although nongraphical browsers, such as Lynx, exist, they are not as popular as graphical browsers, so we will concentrate on the latter. Voice-based browsers are also being developed.

Users who are curious about the Department of Animal Psychology can learn more about it by clicking on its (underlined) name. The browser then fetches the page to which the name is linked and displays it, as shown in Fig. 7-58(b). The underlined items here can also be clicked on to fetch other pages, and so on. The new page can be on the same machine as the first one, or on a machine halfway around the globe. The user cannot tell. Page fetching is done by the browser, without any help from the user. If the user ever returns to the main page, the links that have already been followed may be shown with a dotted underline (and possibly a different color) to distinguish them from links that have not been followed. Note that clicking on the *Campus Information* line in the main page does nothing. It is not underlined, which means that it is just text and is not linked to another page.

Most browsers have numerous buttons and features to make it easier to navigate the Web. Many have a button for going back to the previous page, a button for going forward to the next page (only operative after the user has gone back from it), and a button for going straight to the user's own home page. Most browsers have a button or menu item to set a bookmark on a given page and another one to display the list of bookmarks, making it possible to revisit any of



WELCOME TO THE UNIVERSITY OF EAST PODUNK'S WWW HOME PAGE

- Campus Information
  - ☐ Admissions information
  - ☐ Campus map
  - ☐ Directions to campus
  - ☐ The UEP student body

- Academic Departments
  - ☐ Department of Animal Psychology
  - ☐ Department of Alternative Studies
  - ☐ Department of Microbiotic Cooking
  - ☐ Department of Nontraditional Studies
  - ☐ Department of Traditional Studies

Webmaster@eastpodunk.edu

(a)

THE DEPARTMENT OF ANIMAL PSYCHOLOGY

- Information for prospective majors
- Personnel
  - ☐ Faculty members
  - ☐ Graduate students
  - ☐ Nonacademic staff
- Research Projects
- Positions available
- Our most popular courses
  - ☐ Dealing with herbivores
  - ☐ Horse management
  - ☐ Negotiating with your pet
  - ☐ User-friendly doghouse construction
- Full list of courses

Webmaster@animalpsyc.eastpodunk.edu

(b)

Fig. 7-58. (a) A Web page. (b) The page reached by clicking on Department of Animal Psychology them with a single mouse click. Pages can also be saved to disk or printed. Numerous options are generally available for controlling the screen layout and setting various user preferences. A comparison of nine browsers is given in (Berghel, 1996).

In addition to having ordinary text (not underlined) and hypertext (underlined), Web pages can also contain icons, line drawings, maps, and photographs. Each of these can (optionally) be linked to another page. Clicking on one of these elements causes the browser to fetch the linked page and display it, the same as clicking on text. With images such as photos and maps, which page is fetched next may depend on what part of the image was clicked on.

Not all pages are viewable in the conventional way. For example, some pages consist of audio tracks, video clips, or both. When hypertext pages are mixed with other media, the result is called **hypermedia**. Some browsers can display all kinds of hypermedia, but others cannot. Instead they check a configuration file to see how to handle the received data. Normally, the configuration file gives the name of a program, called an **external viewer**, or a **helper application**, to be run with the incoming page as input. If no viewer is configured, the browser usually asks the user to choose one. If no viewer exists, the user can tell the browser to save the incoming page to a disk file, or to discard it. Helper applications for producing speech are making it possible for even blind users to access the Web. Other helper applications contain interpreters for special Web languages, making it possible to download and run programs from Web pages. This mechanism makes it possible to extend the functionality of the Web itself.

Many Web pages contain large images, which take a long time to load. For example, fetching an uncompressed 640 × 480 (VGA) image with 24 bits per pixel (922 KB) takes about 4 minutes over a 28.8-kbps modem line. Some browsers deal with the slow loading of images by first fetching and displaying the text, then getting the images. This strategy gives the user something to read while the images are coming in and also allows the user to kill the load if the page is not sufficiently interesting to warrant waiting. An alternative strategy is to provide an option to disable the automatic fetching and display of images.

Some page writers attempt to placate potentially bored users by displaying images in a special way. First the image quickly appears in a coarse resolution. Then the details are gradually filled in. For the user, seeing the whole image after a few seconds, albeit at low resolution, is often preferable to seeing it built up slowly from the top, scan line by scan line.

Some Web pages contain forms that request the user to enter information. Typical applications of these forms are searching a database for a user-supplied item, ordering a product, or participating in a public opinion survey. Other Web pages contain maps that allow users to click on them to zoom in or get information about some geographical area. Handling forms and active (clickable) maps requires more sophisticated processing than just fetching a known page. We will describe later how these features are implemented.

Some browsers use the local disk to cache pages that they have fetched. Before a page is fetched, a check is made to see if it is in the local cache. If so, it is only necessary to check if the page if still up to date. If so, the page need not be loaded again. As a result, clicking on the BACK button to see the previous page is normally very fast.

To host a Web browser, a machine must be directly on the Internet, or at least have a SLIP or PPP connection to a router or other machine that is directly on the Internet. This requirement exists because the way a browser fetches a page is to establish a TCP connection to the machine where the page is, and then send a message over the connection asking for the page. If it cannot establish a TCP connection to an arbitrary machine on the Internet, a browser will not work.

Sometimes the lengths that people will go to get Web access are amazing. At least one company is offering Web-by-Fax service. A client without Internet access calls up the Web-by-Fax server and logs in using the telephone keypad. He then types in a code identifying the Web page desired and it is faxed to the caller's fax machine.

7.6.2. The Server Side

Every Web site has a server process listening to TCP port 80 for incoming connections from clients (normally browsers). After a connection has been established, the client sends one request and the server sends one reply. Then the connection is released. The protocol that defines the legal requests and replies is called HTTP. We will study it in some detail below, but a simple example using it may provide a reasonable idea of how Web servers work. Figure 7-59 shows how the various parts of the Web model fit together.

For this example, we can imagine that the user has just clicked on some piece of text or perhaps on an icon that points to the page whose name (URL—Uniform Resource Locator) is *http://www.w3.org/hypertext/WWW/TheProject.html.* We will also explain URLs later on in this chapter. For the moment, it is sufficient to know that a URL has three parts: the name of the protocol (*http*), the name of the machine where the page is located (*www.w3.org*), and the name of the file containing the page (*hypertext/WWW/TheProject.html*). The steps that occur between the user's click and the page being displayed are as follows:

1. The browser determines the URL (by seeing what was selected).
2. The browser asks DNS for the IP address of *www.w3.org*.
3. DNS replies with 18.23.0.23.
4. The browser makes a TCP connection to port 80 on 18.23.0.23.
5. It then sends a *GET /hypertext/WWW/TheProject.html* command.
6. The *www.w3.org* server sends the file *TheProject.html*.

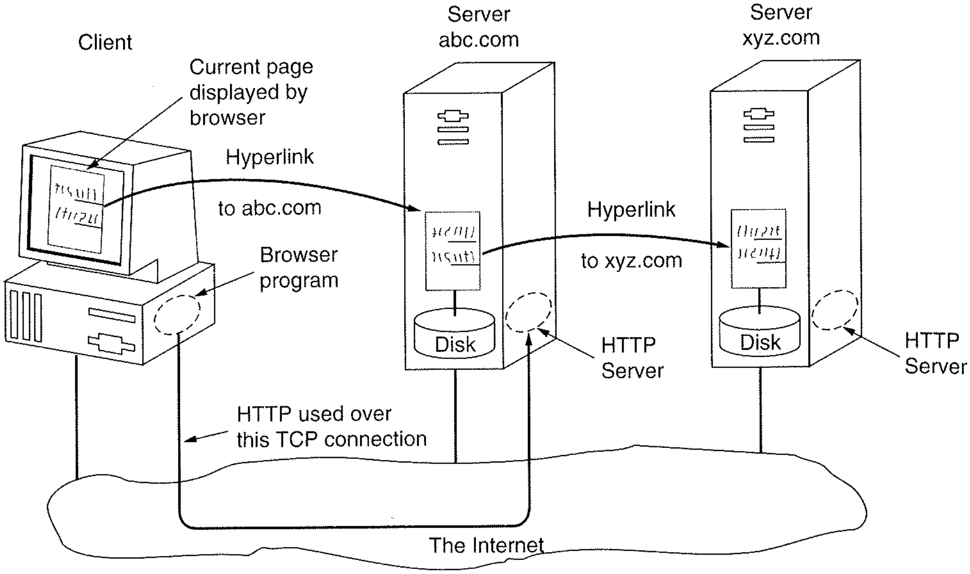


Fig. 7-59. The parts of the Web model.

7. The TCP connection is released.
8. The browser displays all the text in *TheProject.html*.
9. The browser fetches and displays all images in *TheProject.html*.

Many browsers display which step they are currently executing in a status line at the bottom of the screen. In this way, when the performance is poor, the user can see if it is due to DNS not responding, the server not responding, or simply network congestion during page transmission.

It is worth noting that for each in-line image (icon, drawing, photo, etc.) on a page, the browser establishes a new TCP connection to the relevant server to fetch the image. Needless to say, if a page contains many icons, all on the same server, establishing, using, and releasing a new connection for each one is not wildly efficient, but it keeps the implementation simple. Future revisions of the protocol will address the efficiency issue. One proposal is given in (Mogul, 1995).

Because HTTP is an ASCII protocol like SMTP, it is quite easy for a person at a terminal (as opposed to a browser) to directly talk to Web servers. All that is needed is a TCP connection to port 80 on the server. The simplest way to get such a connection is to use the Telnet program. Figure 7-60 shows a scenario of how this can be done. In this example, the lines marked *C:* are typed in by the user (client), the lines marked *T:* are produced by the Telnet program, and the lines marked *S:* are produced by the server at M.I.T.

```
C: telnet  www.w3.org  80
T: Trying 18.23.0.23 ...
T: Connected to www.w3.org.
T: Escape character is '^]'.
C: GET  /hypertext/WWW/TheProject.html  HTTP/1.0
C:
    S: HTTP/1.0 200 Document follows
    S: MIME-Version: 1.0
    S: Server: CERN/3.0
    S: Content-Type: text/html
    S: Content-Length: 8247
    S:
    S: <HEAD> <TITLE> The World Wide Web Consortium (W3C) </TITLE> </HEAD>
    S: <BODY>
    S: <H1> <IMG ALIGN=MIDDLE ALT="W3C" SRC="Icons/WWW/w3c_96x67.gif">
    S: The World Wide Web Consortium </H1> <P>
    S:
    S: The World Wide Web is the universe of network-accessible information.
    S: The <A HREF="Consortium/"> World Wide Web Consortium </A>
    S: exists to realize the full potential of the Web. <P>
    S:
    S: W3C works with the global community to produce
    S: <A HREF="#Specifications"> specifications </A> and
    S: <A HREF="#Reference"> reference software </A> .
    S: W3C is funded by industrial
    S: <A HREF="Consortium/Member/List.html"> members </A>
    S: but its products are freely available to all. <P>
    S:
    S: In this document:
    S: <menu>
    S: <LI> <A HREF="#Specifications"> Web Specifications and Development Areas </A>
    S: <LI> <A HREF="#Reference"> Web Software </A>
    S: <LI> <A HREF="#Community"> The World Wide Web and the Web Community </A>
    S: <LI> <A HREF="#Joining"> Getting involved with the W3C </A>
    S: </menu>
    S: <P> <HR>
    S: <P> W3C is hosted by the
    S: <A HREF="http://www.lcs.mit.edu/"> Laboratory for Computer Science </A> at
    S: <A HREF="http://web.mit.edu/"> MIT </A> , and
    S: in Europe by <A HREF="http://www.inria.fr/"> INRIA </A> .
    S: </BODY>
```

Fig. 7-60. A sample scenario for obtaining a Web page.

Readers are encouraged to try this scenario personally (preferably from a UNIX system, because some other systems do not return the connection status). Be sure to note the spaces and the protocol version on the *GET* line, and the blank line following the *GET* line. As an aside, the actual text that will be received will differ from what is shown in Fig. 7-60 for three reasons. First, the example output here has been abridged and edited to make it fit on one page. Second, it has been cleaned up somewhat to avoid embarrassing the author, who no doubt expected thousands of people to examine the formatted page, but zero people to scrutinize the HTML that produced it. Third, the contents of the page are constantly being revised. Nevertheless, this example should give a reasonable idea of how HTTP works.

What the example shows is the following. The client, in this case a person, but normally a browser, first connects to a particular host and then sends a command asking for a particular page and specifying a particular protocol and version to use (HTTP/1.0). On line 7, the server responds with a status line telling the protocol it is using (the same as the client) and the code 200, meaning OK. This line is followed by an RFC 822 MIME message, of which five of the header lines are shown in the figure (several others have been omitted to save space). Then comes a blank line, followed by the message body. For sending a picture, the *Content-Type* field might be

```
Content-Type: Image/GIF
```

In this way, the MIME types allow arbitrary objects to be sent in a standard way. As an aside, the MIME *Content-Transfer-Encoding* header is not needed because TCP allows arbitrary byte streams, even pictures, to be sent without modification. The meaning of the commands within angle brackets used in the sample page will be discussed later in this chapter.

Not all servers speak HTTP. In particular, many older servers use the FTP, Gopher, or other protocols. Since a great deal of useful information is available on FTP and Gopher servers, one of the design goals of the Web was to make this information available to Web users. One solution is to have the browser use these protocols when speaking to an FTP or Gopher server. Some of them, in fact, use this solution, but making browsers understand every possible protocol makes them unnecessarily large.

Instead, a different solution is often used: proxy servers (Luotonen and Altis, 1994). A **proxy server** is a kind of gateway that speaks HTTP to the browser but FTP, Gopher, or some other protocol to the server. It accepts HTTP requests and translates them into, say, FTP requests, so the browser does not have to understand any protocol except HTTP. The proxy server can be a program running on the same machine as the browser, but it can also be on a free-standing machine somewhere in the network serving many browsers. Figure 7-61 shows the difference between a browser that can speak FTP and one that uses a proxy.

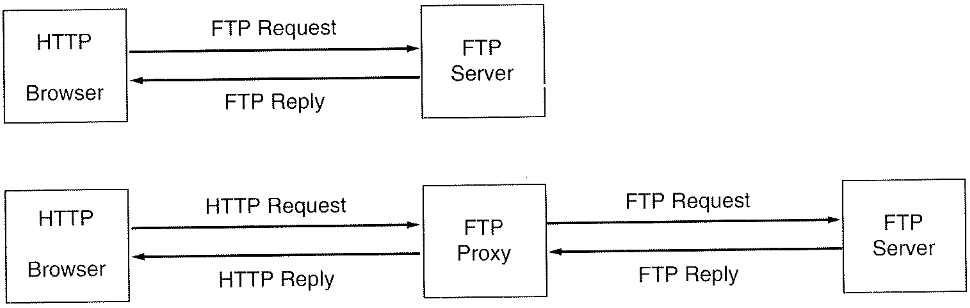


Fig. 7-61. (a) A browser that speaks FTP. (b) A browser that does not.

Often users can configure their browsers with proxies for protocols that the browsers do not speak. In this way, the range of information sources to which the browser has access is increased.

In addition to acting as a go-between for unknown protocols, proxy servers have a number of other important functions, such as caching. A caching proxy server collects and keeps all the pages that pass through it. When a user asks for a page, the proxy server checks to see if it has the page. If so, it can check to see if the page is still current. In the event that the page is still current, it is passed to the user. Otherwise, a new copy is fetched.

Finally, an organization can put a proxy server inside its firewall to allow users to access the Web, but without giving them full Internet access. In this configuration, users can talk to the proxy server, but it is the proxy server that contacts remote sites and fetches pages on behalf of its clients. This mechanism can be used, for example, by high schools, to block access to Web sites the principal feels are inappropriate for tender young minds.

For information about one of the more popular Web servers (NCSA's HTTP daemon) and its performance, see (Katz et al., 1994; and Kwan et al., 1995).

HTTP—HyperText Transfer Protocol

The standard Web transfer protocol is **HTTP (HyperText Transfer Protocol)**. Each interaction consists of one ASCII request, followed by one RFC 822 MIME-like response. Although the use of TCP for the transport connection is very common, it is not formally required by the standard. If ATM networks become reliable enough, the HTTP requests and replies could be carried in AAL 5 messages just as well.

HTTP is constantly evolving. Several versions are in use and others are under development. The material presented below is relatively basic and is unlikely to change in concept, but some details may be a little different in future versions.

The HTTP protocol consists of two fairly distinct items: the set of requests from browsers to servers and the set of responses going back the other way. We will now treat each of these in turn.

All the newer versions of HTTP support two kinds of requests: simple requests and full requests. A simple request is just a single *GET* line naming the page desired, without the protocol version. The response is just the raw page, with no headers, no MIME, and no encoding. To see how this works, try making a Telnet connection to port 80 of *www.w3.org* (as shown in the first line of Fig. 7-60) and then type

```
GET /hypertext/WWW/TheProject.html
```

but without the HTTP/1.0 this time. The page will be returned with no indication of its content type. This mechanism is needed for backward compatibility. Its use will decline as browsers and servers based on full requests become standard.

Full requests are indicated by the presence of the protocol version on the *GET* request line, as in Fig. 7-60. Requests may consist of multiple lines, followed by a blank line to indicate the end of the request, which is why the blank line was needed in Fig. 7-60. The first line of a full request contains the command (of which *GET* is but one of the possibilities), the page desired, and the protocol/version. Subsequent lines contain RFC 822 headers.

Although HTTP was designed for use in the Web, it has been intentionally made more general than necessary with an eye to future object-oriented applications. For this reason, the first word on the full request line is simply the name of the **method** (command) to be executed on the Web page (or general object). The built-in methods are listed in Fig. 7-62. When accessing general objects, additional object-specific methods may also be available. The names are case sensitive, so, *GET* is a legal method but *get* is not.

Method	Description
GET	Request to read a Web page
HEAD	Request to read a Web page's header
PUT	Request to store a Web page
POST	Append to a named resource (e.g., a Web page)
DELETE	Remove the Web page
LINK	Connects two existing resources
UNLINK	Breaks an existing connection between two resources

Fig. 7-62. The built-in HTTP request methods.

The *GET* method requests the server to send the page (by which we mean object, in the most general case), suitably encoded in MIME. However, if the

GET request is followed by an *If-Modified-Since* header, the server only sends the data if it has been modified since the date supplied. Using this mechanism, a browser that is asked to display a cached page can conditionally ask for it from the server, giving the modification time associated with the page. If the cache page is still valid, the server just sends back a status line announcing that fact, thus eliminating the overhead of transferring the page again.

The *HEAD* method just asks for the message header, without the actual page. This method can be used to get a page's time of last modification, to collect information for indexing purposes, or just to test a URL for validity. Conditional *HEAD* requests do not exist.

The *PUT* method is the reverse of *GET*: instead of reading the page, it writes the page. This method makes it possible to build a collection of Web pages on a remote server. The body of the request contains the page. It may be encoded using MIME, in which case the lines following the *PUT* might include *Content-Type* and authentication headers, to prove that the caller indeed has permission to perform the requested operation.

Somewhat similar to *PUT* is the *POST* method. It too bears a URL, but instead of replacing the existing data, the new data is "appended" to it in some generalized sense. Posting a message to a news group or adding a file to a bulletin board system are examples of appending in this context. It is clearly the intention here to have the Web take over the functionality of the USENET news system.

DELETE does what you might expect: it removes the page. As with *PUT*, authentication and permission play a major role here. There is no guarantee that *DELETE* succeeds, since even if the remote HTTP server is willing to delete the page, the underlying file may have a mode that forbids the HTTP server from modifying or removing it.

The *LINK* and *UNLINK* methods allow connections to be established between existing pages or other resources.

Every request gets a response consisting of a status line, and possibly additional information (e.g., all or part of a Web page). The status line can bear the code 200 (OK), or any one of a variety of error codes, for example 304 (not modified), 400 (bad request), or 403 (forbidden).

The HTTP standards describe message headers and bodies in considerable detail. Suffice it to say that these are very close to RFC 822 MIME messages, so we will not look at them here.

7.6.3. Writing a Web Page in HTML

Web pages are written in a language called **HTML (HyperText Markup Language)**. HTML allows users to produce Web pages that include text, graphics, and pointers to other Web pages. We will begin our study of HTML with these pointers, since they are the glue that holds the Web together.

URLs—Uniform Resource Locators

We have repeatedly said that Web pages may contain pointers to other Web pages. Now it is time to see how these pointers are implemented. When the Web was first created, it was immediately apparent that having one page point to another Web page required mechanisms for naming and locating pages. In particular, there were three questions that had to be answered before a selected page could be displayed:

1. What is the page called?
2. Where is the page located?
3. How can the page be accessed?

If every page were somehow assigned a unique name, there would not be any ambiguity in identifying pages. Nevertheless, the problem would not be solved. Consider a parallel between people and pages. In the United States, almost everyone has a social security number, which is a unique identifier, as no two people have the same one. Nevertheless, armed only with a social security number, there is no way to find the owner's address, and certainly no way to tell whether you should write to the person in English, Spanish, or Chinese. The Web has basically the same problems.

The solution chosen identifies pages in a way that solves all three problems at once. Each page is assigned a **URL** (**Uniform Resource Locator**) that effectively serves as the page's worldwide name. URLs have three parts: the protocol (also called a scheme), the DNS name of the machine on which the page is located, and a local name uniquely indicating the specific page (usually just a file name on the machine where it resides). For example, the URL for the author's department is

http://www.cs.vu.nl/welcome.html

This URL consists of three parts: the protocol (*http*), the DNS name of the host (*www.cs.vu.nl*), and the file name (*welcome.html*), with certain punctuation separating the pieces.

Many sites have certain shortcuts for file names built in. For example, *~user/* might be mapped onto *user*'s WWW directory, with the convention that a reference to the directory itself implies a certain file, say, *index.html*. Thus the author's home page can be reached at

http://www.cs.vu.nl/~ast/

even though the actual file name is different. At many sites, a null file name defaults to the organization's home page.

Now it should be clear how hypertext works. To make a piece of text clickable, the page writer must provide two items of information: the clickable text to be displayed and the URL of the page to go to if the text is selected. When the text is selected, the browser looks up the host name using DNS. Now armed with the host's IP address, the browser then establishes a TCP connection to the host. Over that connection, it sends the file name using the specified protocol. Bingo. Back comes the page. This is precisely what we saw in Fig. 7-60.

This URL scheme is open-ended in the sense that it is straightforward to have protocols other than HTTP. In fact, URLs for various other common protocols have been defined, and many browsers understand them. Slightly simplified forms of the more common ones are listed in Fig. 7-63.

Name	Used for	Example
http	Hypertext (HTML)	http://www.cs.vu.nl/~ast/
ftp	FTP	ftp://ftp.cs.vu.nl/pub/minix/README
file	Local file	/usr/suzanne/prog.c
news	News group	news:comp.os.minix
news	News article	news:AA0134223112@cs.utah.edu
gopher	Gopher	gopher://gopher.tc.umn.edu/11/Libraries
mailto	Sending email	mailto:kim@acm.org
telnet	Remote login	telnet://www.w3.org:80

Fig. 7-63. Some common URLs.

Let us briefly go over the list. The *http* protocol is the Web's native language, the one spoken by HTTP servers. It supports all the methods of Fig. 7-62, as well as whatever object-specific methods are needed.

The *ftp* protocol is used to access files by FTP, the Internet's file transfer protocol. FTP has been around more than two decades and is well entrenched. Numerous FTP servers all over the world allow people anywhere on the Internet to log in and download whatever files have been placed on the FTP server. The Web does not change this; it just makes obtaining files by FTP easier, as FTP has a somewhat arcane interface. In due course, FTP will probably vanish, as there is no particular advantage for a site to run an FTP server instead of an HTTP server, which can do everything that the FTP server can do, and more (although there are some arguments about efficiency).

It is possible to access a local file as a Web page, either by using the *file* protocol, or more simply, by just naming it. This approach is similar to using FTP but does not require having a server. Of course, it only works for local files.

The *news* protocol allows a Web user to call up a news article as though it were a Web page. This means that a Web browser is simultaneously a news reader. In fact, many browsers have buttons or menu items to make reading USENET news even easier than using standard news readers.

Two formats are supported for the *news* protocol. The first format specifies a newsgroup and can be used to get a list of articles from a preconfigured news site. The second one requires the identifier of a specific news article to be given, in this case *AA0134223112@cs.utah.edu*. The browser then fetches the given article from its preconfigured news site using the NNTP protocol.

The *gopher* protocol is used by the Gopher system, which was designed at the University of Minnesota and named after the school's athletic teams, the Golden Gophers (as well as being a slang expression meaning "go for", i.e., go fetch). Gopher predates the Web by several years. It is an information retrieval scheme, conceptually similar to the Web itself, but supporting only text and no images. When a user logs into a Gopher server, he is presented with a menu of files and directories, any of which can be linked to another Gopher menu anywhere in the world.

Gopher's big advantage over the Web is that it works very well with 25 × 80 ASCII terminals, of which there are still quite a few around, and because it is text based, it is very fast. Consequently, there are thousands of Gopher servers all over the world. Using the *gopher* protocol, Web users can access Gopher and have each Gopher menu presented as a clickable Web page. If you are not familiar with Gopher, try the example given in Fig. 7-63 or have your favorite Web search engine look for "gopher."

Although the example given does not illustrate it, it is also possible to send a complete query to a Gopher server using the *gopher+* protocol. What is displayed is the result of querying the remote Gopher server.

The last two protocols do not really have the flavor of fetching Web pages, and are not supported by all browsers, but are useful anyway. The *mailto* protocol allows users to send email from a Web browser. The way to do this is to click on the OPEN button and specify a URL consisting of *mailto:* followed by the recipient's email address. Most browsers will respond by popping up a form containing slots for the subject and other header lines and space for typing the message.

The *telnet* protocol is used to establish an on-line connection to a remote machine. It is used the same way as the Telnet program, which is not surprising, since most browsers just call the Telnet program as a helper application. As an exercise, try the scenario of Fig. 7-60 again, but now using a Web browser.

In short, the URLs have been designed to not only allow users to navigate the Web, but to deal with FTP, news, Gopher, email, and telnet as well, making all the specialized user interface programs for those other services unnecessary, and thus integrating nearly all Internet access into a single program, the Web browser. If it were not for the fact that this scheme was designed by a physics researcher, it could easily pass for the output of some software company's advertising department.

Despite all these nice properties, the growing use of the Web has turned up an inherent weakness in the URL scheme. A URL points to one specific host. For pages that are heavily referenced, it is desirable to have multiple copies far apart, to reduce the network traffic. The trouble is that URLs do not provide any way to reference a page without simultaneously telling where it is. There is no way to say: "I want page xyz, but I do not care where you get it." To solve this problem and make it possible to replicate pages, the IETF is working on a system of **URIs** (**Universal Resource Identifiers**). A URI can be thought of as a generalized URL. This topic is the subject of much current research.

Although we have discussed only absolute URLs here, relative URLs also exist. The difference is analogous to the difference between the absolute file name */usr/ast/foobar* and just *foobar* when the context is unambiguously defined.

HTML—HyperText Markup Language

Now that we have a good idea of how URLs work, it is time to look at HTML itself. HTML is an application of ISO standard 8879, **SGML** (**Standard Generalized Markup Language**), but specialized to hypertext and adapted to the Web.

As mentioned earlier, HTML is a markup language, a language for describing how documents are to be formatted. The term "markup" comes from the old days when copyeditors actually marked up documents to tell the printer—in those days, a human being—which fonts to use, and so on. Markup languages thus contain explicit commands for formatting. For example, in HTML, <B> **means start boldface mode, and** </B> means leave boldface mode. The advantage of a markup language over one with no explicit markup is that writing a browser for it is straightforward: the browser simply has to understand the markup commands. TeX and troff are other well-known examples of markup languages.

Documents written in a markup language can be contrasted to documents produced with a WYSIWYG (What You See Is What You Get) word processor, such as MS-Word® or WordPerfect®. These systems may store their files with hidden embedded markup so they can reproduce them later, but not all of them work this way. Word processors for the Macintosh, for example, keep the formatting information in separate data structures, not as commands embedded in the user files.

By embedding the markup commands within each HTML file and standardizing them, it becomes possible for any Web browser to read and reformat any Web page. Being able to reformat Web pages after receiving them is crucial because a page may have been produced full screen on a 1024 × 768 display with 24-bit color but may have to be displayed in a small window on a 640 × 480 screen with 8-bit color. Proprietary WYSIWYG word processors cannot be used on the Web because their internal markup languages (if any) are not standardized across vendors, machines and operating systems. Also, they do not handle reformatting for different-sized windows and different resolution displays. However, word processing programs can offer the option of saving documents in HTML instead of in the vendor's proprietary format, and some of them already do.

Like HTTP, HTML is in a constant state of flux. When Mosaic was the only browser, the language it interpreted, HTML 1.0, was the de facto standard. When new browsers came along, there was a need for a formal Internet standard, so the HTML 2.0 standard was produced. HTML 3.0 was initially created as a research effort to add many new features to HTML 2.0, including tables, toolbars, mathematical formulas, advanced style sheets (for defining page layout and the meaning of symbols), and more.

The official standardization of HTML is being managed by the WWW Consortium, but various browser vendors have added their own ad hoc extensions. These vendors hope to get people to write Web pages using their extensions, so readers of these pages will need the vendor's browser to properly interpret the pages. This tendency does not make HTML standardization any easier.

Below we will give a brief introduction to HTML, just to give an idea of what it is like. While it is certainly possible to write HTML documents with any standard editor, and many people do, it is also possible to use special HTML editors that do most of the work (but correspondingly give the user less control over all the details of the final result).

A proper Web page consists of a head and a body enclosed by <HTML> and </HTML> **tags** (formatting commands), although most browsers do not complain if these tags are missing. As can be seen from Fig. 7-64(a), the head is bracketed by the <HEAD> and </HEAD> tags and the body is bracketed by the <BODY> and </BODY> tags. The commands inside the tags are called **directives**. Most HTML tags have this format, that is, <SOMETHING> to mark the beginning of something and </SOMETHING> to mark its end. Numerous other examples of HTML are easily available. Most browsers have a menu item VIEW SOURCE or something like that. Selecting this item displays the current page's HTML source, instead of its formatted output.

Tags can be in either lowercase or uppercase. Thus <HEAD> and <head> mean the same thing, but the former stands out better for human readers. Actual layout of the HTML document is irrelevant. HTML parsers ignore extra spaces and carriage returns since they have to reformat the text to make it fit the current display area. Consequently, white space can be added at will to make HTML documents more readable, something most of them are badly in need of. As another consequence, blank lines cannot be used to separate paragraphs, as they are simply ignored. An explicit tag is required.

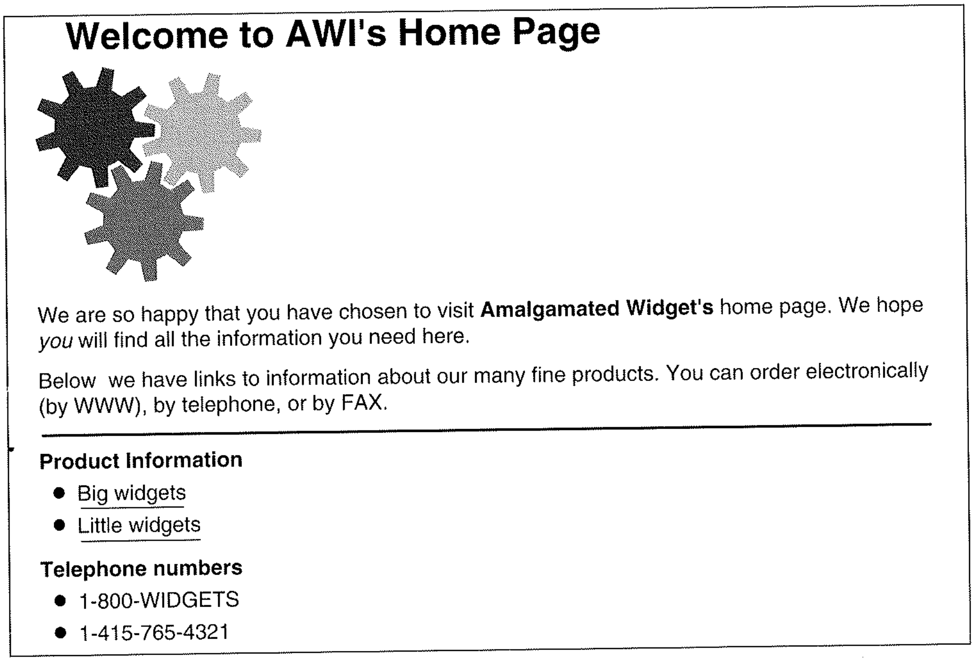
Some tags have (named) parameters. For example

```
<IMG SRC="abc"  ALT="foobar">
```

is a tag, <IMG>, with parameter *SRC* set equal to *abc* and parameter *ALT* set equal to *foobar*. For each tag, the HTML standard gives a list of what the permitted parameters, if any, are, and what they mean. Because each parameter is named, the order in which the parameters are given is not significant.

```
<HTML> <HEAD> <TITLE> AMALGAMATED WIDGET, INC. </TITLE> </HEAD>
<BODY> <H1> Welcome to AWI's Home Page </H1>
<IMG SRC="http://www.widget.com/images/logo.gif" ALT="AWI Logo"> <BR>
We are so happy that you have chosen to visit <B> Amalgamated Widget's</B>
home page. We hope <I> you </I> will find all the information you need here.
<P>Below we have links to information about our many fine products.
You can order electronically (by WWW), by telephone, or by fax. <HR>
<H2> Product information </H2>
<UL> <LI> <A HREF="http://widget.com/products/big">  Big widgets    </A>
        <LI> <A HREF="http://widget.com/products/little"> Little widgets </A>
</UL>
<H2> Telephone numbers </H2>
<UL> <LI> By telephone: 1-800-WIDGETS
        <LI> By fax: 1-415-765-4321
</UL> </BODY> </HTML>
```

(a)



Welcome to AWI's Home Page

We are so happy that you have chosen to visit **Amalgamated Widget's** home page. We hope *you* will find all the information you need here.

Below we have links to information about our many fine products. You can order electronically (by WWW), by telephone, or by FAX.

Product Information

- Big widgets
- Little widgets

Telephone numbers

- 1-800-WIDGETS
- 1-415-765-4321

(b)

Fig. 7-64. (a) The HTML for a sample Web page. (b) The formatted page.

Technically, HTML documents are written in the ISO 8859-1 Latin-1 character set, but for users whose keyboards only support ASCII, escape sequences are present for the special characters, such as è. The list of special characters is given in the standard. All of them begin with an ampersand and end with a semicolon. For example, è produces è and é produces é. Since <, >, and & have special meanings, they can be expressed only with their escape sequences, < > and & respectively.

The main item in the head is the title, delimited by <TITLE> and </TITLE>, but certain kinds of meta-information may also be present. The title itself is not displayed on the page. Some browsers use it to label the page's window.

Let us now take a look at some of the other features illustrated in Fig. 7-64. All of the tags used in Fig. 7-64 and some others are shown in Fig. 7-65. Headings are generated by an <H*n*> tag, where *n* is a digit in the range 1 to 6. <H1> is the most important heading; <H6> is the least important one. It is up to the browser to render these appropriately on the screen. Typically the lower numbered headings will be displayed in a larger and heavier font. The browser may also choose to use different colors for each level of heading. Typically <H1> headings are large and boldface with at least one blank line above and below. In contrast, <H2> headings are in a smaller font, and with less space above and below.

The tags <B> and <I> are used to enter boldface and italics mode, respectively. If the browser is not capable of displaying boldface and italics, it must use some other method of rendering them, for example, using a different color for each or perhaps reverse video. Instead of specifying physical styles such as boldface and italics, authors can also use logical styles such as <DN> (define), <EM> (weak emphasis), <STRONG> (strong emphasis), and <VAR> (program variables). The logical styles are defined in a document called a **style sheet**. The advantage of the logical styles is that by changing one definition, all the variables can be changed, for example, from italics to a constant width font.

HTML provides various mechanisms for making lists, including nested lists. The <UL> tag starts an unordered list. The individual items, which are marked with the <LI> tag in the source, appear with bullets (•) in front of them. A variant of this mechanism is <OL>, which is for ordered lists. When this tag is used, the <LI> items are numbered by the browser. A third option is <MENU>, which typically produces a more compact list on the screen, with no bullets and no numbers. Other than the use of different starting and ending tags, <UL>, <OL>, and <MENU> have the same syntax and similar results.

In addition to the list mechanisms shown in Fig. 7-65, there are two others that are worth mentioning briefly. <DIR> can be used for making short tables. Also, <DL> and </DL> can make definition lists (glossaries) with two-part entries, whose parts are defined by <DT> and <DD> respectively. The first is for the name, the second for its meaning. These features are largely superseded by the (more general and complex) table mechanism, described below.

Tag	Description
<HTML> ... </HTML>	Declares the Web page to be written in HTML
<HEAD> ... </HEAD>	Delimits the page's head
<TITLE> ... </TITLE>	Defines the title (not displayed on the page)
<BODY> ... </BODY>	Delimits the page's body
<H*n*> ... </H*n*>	Delimits a level *n* heading
<B> ... </B>	Set ... in boldface
<I> ... </I>	Set ... in italics
<UL> ... </UL>	Brackets an unordered (bulleted) list
<OL> ... </OL>	Brackets a numbered list
<MENU> ... </MENU>	Brackets a menu of <LI> items
<LI>	Start of a list item (there is no </LI>)
 	Force a break here
<P>	Start of paragraph
<HR>	Horizontal rule
<PRE> ... </PRE>	Preformatted text; do not reformat
<IMG SRC="...">	Load an image here
<A HREF="..."> ... </A>	Defines a hyperlink

Fig. 7-65. A selection of common HTML tags. Some have additional parameters.

The <BR>, <P>, and <HR> tags all indicate a boundary between sections of text. The precise format can be determined by the style sheet associated with the page. The <BR> tag just forces a line break. Typically, browsers do not insert a blank line after <BR>. In contrast, <P> starts a paragraph, which might, for example, insert a blank line and possibly some indentation. (Theoretically, </P> exists to mark the end of a paragraph, but it is rarely used; most HTML authors do not even know it exists.) Finally, <HR> forces a break and draws a horizontal line across the screen.

HTML 1.0 had no ability to display tables or other formatted information. Worse yet, if the HTML writer carefully formatted a table by judicious use of spaces and carriage returns, browsers would ignore all the layout and display the page as if all the formatted material were unformatted. To prevent browsers from messing up carefully laid out text, the <PRE> and </PRE> tags were provided. They are instructions to the browser to just display everything in between literally, character for character, without changing anything. As the table and other fancy layout features become more widely implemented, the need for <PRE> will diminish, except for program listings, for which most programmers will tolerate no formatting other than their own.

HTML allows images to be included in-line on a Web page. The <IMG> tag specifies that an image is to be loaded at the current position in the page. It can have several parameters. The *SRC* parameter gives the URL (or URI) of the image. The HTML standard does not specify which graphic formats are permitted. In practice, all browsers support GIF files and many support JPEG files as well. Browsers are free to support other formats, but this extension is a two-edged sword. If a user is accustomed to a browser that supports, say, BMP files, he may include these in his Web pages and later be surprised when other browsers just ignore all of his wonderful art.

Other parameters of <IMG> are *ALIGN*, which controls the alignment of the image with respect to the text baseline (*TOP*, *MIDDLE*, *BOTTOM*), *ALT*, which provides text to use instead of the image when the user has disabled images, and *ISMAP*, a flag indicating that the image is an active map.

Finally, we come to hyperlinks, which use the <A> (anchor) and </A> tags. Like <IMG>, <A> has various parameters, including *HREF* (the URL), *NAME* (the hyperlink's name), and *METHODS* (access methods), among others. The text between the <A> and </A> is displayed. If it is selected, the hyperlink is followed to a new page. It is also permitted to put an <IMG> image there, in which case clicking on the image also activates the hyperlink.

As an example, consider the following HTML fragment:

```
<A HREF="http://www.nasa.gov"> NASA's home page </A>
```

When a page with this fragment is displayed, what appears on the screen is

NASA's home page

If the user subsequently clicks on this text, the browser immediately fetches the page whose URL is *http://www.nasa.gov* and displays it.

As a second example, now consider

```
<A HREF="http://www.nasa.gov"> <IMG SRC="shuttle.gif" ALT="NASA"> </A>
```

When displayed, this page shows a picture (e.g., of the space shuttle). Clicking on the picture switches to NASA's home page, just as clicking on the underlined text did in the previous example. If the user has disabled automatic image display, the text NASA will be displayed where the picture belongs.

The <A> tag can take a parameter *NAME* to plant a hyperlink, so it can be referred to from within the page. For example, some Web pages start out with a clickable table of contents. By clicking on an item in the table of contents, the user jumps to the corresponding section of the page.

One feature that HTML 2.0 did not include and which many page authors missed, was the ability to create tables whose entries could be clicked on to active hyperlinks. As a consequence, a large amount of work was done to add tables to

HTML 3.0. Below we give a very brief introduction to tables, just to capture the essential flavor.

An HTML table consists of one or more rows, each consisting of one or more **cells**. Cells can contain a wide range of material, including text, figures, and even other tables. Cells can be merged, so, for example, a heading can span multiple columns. Page authors have limited control over the layout, including alignment, border styles, and cell margins, but the browsers have the final say in rendering tables.

An HTML table definition is listed in Fig. 7-66(a) and a possible rendition is shown in Fig. 7-66(b). This example just shows a few of the basic features of HTML tables. Tables are started by the <TABLE> tag. Additional information can be provided to describe general properties of the table.

The <CAPTION> tag can be used to provide a figure caption. Each row is started with a <TR> (Table Row) tag. The individual cells are marked as <TH> (Table Header) or <TD> (Table Data). The distinction is made to allow browsers to use different renditions for them, as we have done in the example.

Numerous other tags are also allowed in tables. They include ways to specify horizontal and vertical cell alignments, justification within a cell, borders, grouping of cells, units, and more.

Forms

HTML 1.0 was basically one way. Users could call up pages from information providers, but it was difficult to send information back the other way. As more and more commercial organizations began using the Web, there was a large demand for two-way traffic. For example, many companies wanted to be able to take orders for products via their Web pages, software vendors wanted to distribute software via the Web and have customers fill out their registration cards electronically, and companies offering Web searching wanted to have their customers be able to type in search keywords.

These demands led to the inclusion of **forms** starting in HTML 2.0. Forms contain boxes or buttons that allow users to fill in information or make choices and then send the information back to the page's owner. They use the <INPUT> tag for this purpose. It has a variety of parameters for determining the size, nature, and usage of the box displayed. The most common forms are blank fields for accepting user text, boxes that can be checked, active maps, and SUBMIT buttons. The example of Fig. 7-67 illustrates some of these choices.

Let us start our discussion of forms by going over this example. Like all forms, this one is enclosed between the <FORM> and </FORM> tags. Text not enclosed in a tag is just displayed. All the usual tags (e.g., <B>) are allowed in a form. Three kinds of input boxes are used in this form.

The first kind of input box follows the text "Name". The box is 46 characters wide and expects the user to type in a string, which is then stored in the variable

```
<HTML> <HEAD> <TITLE> A sample page with a table </TITLE> </HEAD>
<BODY>
<TABLE BORDER=ALL RULES=ALL>
<CAPTION> Some Differences between HTML Versions </CAPTION>
<COL ALIGN=LEFT>
<COL ALIGN=CENTER>
<COL ALIGN=CENTER>
<COL ALIGN=CENTER>
<TR> <TH>Item  <TH>HTML 1.0  <TH>HTML 2.0  <TH>HTML 3.0
<TR> <TH> Active Maps and Images <TD> <TD> x <TD> x
<TR> <TH> Equations <TD> <TD> <TD> x
<TR> <TH> Forms <TD> <TD> x <TD> x
<TR> <TH> Hyperlinks x <TD> <TD> x <TD> x
<TR> <TH> Images <TD> x <TD> x <TD> x
<TR> <TH> Lists <TD> x <TD> x <TD> x
<TR> <TH> Toolbars <TD> <TD> <TD> x
<TR> <TH> Tables <TD> <TD> <TD> x
</TABLE> </BODY> </HTML>
```

(a)

Some Differences between HTML Versions

Item	**HTML 1.0**	**HTML 2.0**	**HTML 3.0**
Active Maps and Images		x	x
Equations			x
Forms		x	x
Hyperlinks	x	x	x
Images	x	x	x
Lists	x	x	x
Toolbars			x
Tables			x

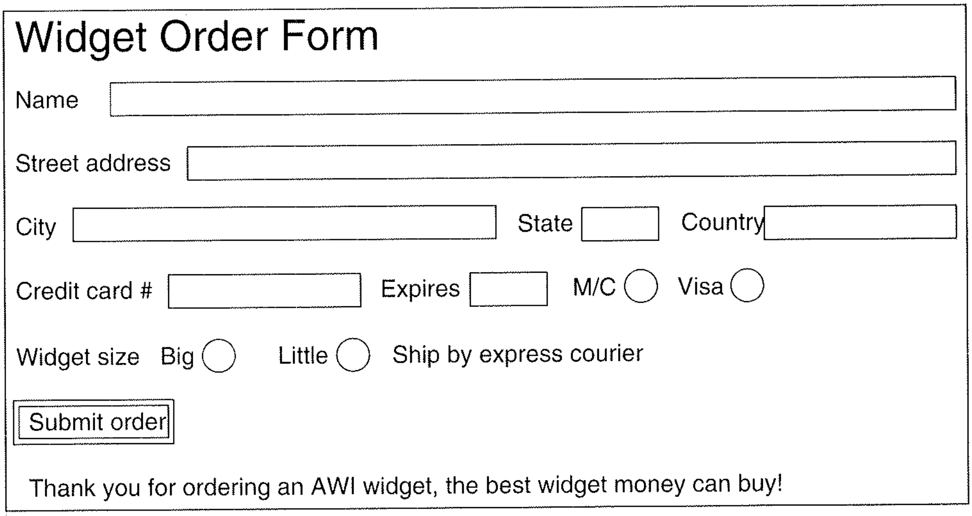
Fig. 7-66. (a) An HTML table. (b) A possible rendition of this table.

customer for later processing. The <P> tag instructs the browser to display subsequent text and boxes on the next line, even if there is room on the current line. By using <P> and other layout tags, the author of the page can control the look of the form on the screen.

The next line of the form asks for the user's street address, 40 columns wide, also on a line by itself. Then comes a line asking for the city, state, and country. No <P> tags are used between the fields here, so the browser displays them all on one line if they will fit. As far as the browser is concerned, this paragraph just contains six items: three strings alternating with three boxes. It displays them linearly from left to right, going over to a new line whenever the current line

```
<HTML> <HEAD> <TITLE> AWI CUSTOMER ORDERING FORM </TITLE> </HEAD>
<BODY>
<H1> Widget Order Form </H1>
<FORM ACTION="http://widget.com/cgi-bin/widgetorder" METHOD=POST>
Name <INPUT NAME="customer" SIZE=46> <P>
Street Address <INPUT NAME="address" SIZE=40> <P>
City <INPUT NAME="city" SIZE=20> State <INPUT NAME="state" SIZE =4>
Country <INPUT NAME="country" SIZE=10> <P>
Credit card # <INPUT NAME="cardno" SIZE=10>
Expires <INPUT NAME="expires" SIZE=4>
M/C <INPUT NAME="cc" TYPE=RADIO VALUE="mastercard">
VISA <INPUT NAME="cc" TYPE=RADIO VALUE="visacard"> <P>
Widget size Big <INPUT NAME="product" TYPE=RADIO VALUE="expensive">
Little <INPUT NAME="product" TYPE=RADIO VALUE="cheap">
Ship by express courier <INPUT NAME="express" TYPE=CHECKBOX> <P>
<INPUT TYPE=SUBMIT VALUE="Submit order"> <P>
Thank you for ordering an AWI widget, the best widget money can buy!
</FORM> </BODY> </HTML>
```

(a)



Widget Order Form

Name

Street address

City State Country

Credit card # Expires M/C ○ Visa ○

Widget size Big ○ Little ○ Ship by express courier

Submit order

Thank you for ordering an AWI widget, the best widget money can buy!

(b)

Fig. 7-67. (a) The HTML for an order form. (b) The formatted page.

cannot hold the next item. Thus it is conceivable that on a 1024 × 768 screen all three strings and their corresponding boxes will appear on the same line, but on a 640 × 480 screen they might be split over two lines. In the worst scenario, the word "Country" is at the end of one line and its box is at the beginning of the next line. There is no way to tell the browser to force the box adjacent to the text.

The next line asks for the credit card number and expiration date. Transmitting credit card numbers over the Internet should only be done when adequate security measures have been taken. For example, some, but not all, browsers encrypt information sent by users. Even then, secure communication and key management are complicated matters and are subject to many kinds of attacks, as we saw earlier.

Following the expiration date we encounter a new feature: radio buttons. These are used when a choice must be made among two or more alternatives. The intellectual model here is a car radio with half a dozen buttons for choosing stations. The browser displays these boxes in a form that allows the user to select and deselect them by clicking on them (or using the keyboard). Clicking on one of them turns off all the other ones in the same group. The visual presentation depends on the graphical interface being used. It is up to the browser to choose a form that is consistent with Windows, Motif, OS/2 Warp, or whatever windowing system is being used. The widget size also uses two radio buttons. The two groups are distinguished by their *NAME* field, not by static scoping using something like <RADIOBUTTON> ... </RADIOBUTTON>.

The *VALUE* parameters are used to indicate which radio button was pushed. Depending on which of the credit card options the user has chosen, the variable *cc* will be set to either the string "mastercard" or the string "visacard".

After the two sets of radio buttons, we come to the shipping option, represented by a box of type *CHECKBOX*. It can be either on or off. Unlike radio buttons, where exactly one out of the set must be chosen, each box of type *CHECKBOX* can be on or off, independently of all the others. For example, when ordering a pizza via Electropizza's Web page, the user can choose sardines *and* onions *and* pineapple (if she can stand it), but she cannot choose small *and* medium *and* large for the same pizza. The pizza toppings would be represented by three separate boxes of type *CHECKBOX*, whereas the pizza size would be a set of radio buttons.

As an aside, for very long lists from which a choice must be made, radio buttons are somewhat inconvenient. Therefore, the <SELECT> and </SELECT> tags are provided to bracket a list of alternatives, but with the semantics of radio buttons (unless the *MULTIPLE* parameter is given, in which case the semantics are those of checkable boxes). Some browsers render the items between <SELECT> and </SELECT> as a pop-up menu.

We have now seen two of the built-in types for the <INPUT> tag: *RADIO* and *CHECKBOX*. In fact, we have already seen a third one as well: *TEXT*. Because this type is the default, we did not bother to include the parameter *TYPE = TEXT*, but we could have. Two other types are *PASSWORD* and *TEXTAREA*. A *PASSWORD* box is the same as a *TEXT* box, except that the characters are not displayed as they are typed. A *TEXTAREA* box is also the same as a *TEXT* box, except that it can contain multiple lines.

Getting back to the example of Fig. 7-67, we now come across an example of a *SUBMIT* button. When this is clicked, the user information on the form is sent back to the machine that provided the form. Like all the other types, *SUBMIT* is a reserved word that the browser understands. The *VALUE* string here is the label on the button and is displayed. All boxes can have values; we only needed that feature here. For *TEXT* boxes, the contents of the *VALUE* field are displayed along with the form, but the user can edit or erase it. *CHECKBOX* and *RADIO* boxes can also be initialized, but with a field called *CHECKED* (because *VALUE* just gives the text, but does not indicate a preferred choice).

The browser also understands the *RESET* button. When clicked, it resets the form to its initial state.

Two more types are worth noting. The first is the *HIDDEN* type. This is output only; it cannot be clicked or modified. For example, when working through a series of pages throughout which choices have to be made, previously made choices might be of *HIDDEN* type, to prevent them from being changed.

Our last type is *IMAGE*, which is for active maps (and other clickable images). When the user clicks on the map, the (x, y) coordinates of the pixel selected (i.e., the current mouse position) are stored in variables and the form is automatically returned to the owner for further processing.

Forms can be submitted in three ways: using the submit button, clicking on an active map, or typing ENTER on a one-item *TEXT* form. When a form is submitted, some action must be taken. The action is specified by the parameters of the <FORM> tag. The *ACTION* parameter specifies the URL (or URI) to tell about the submission, and the *METHOD* parameter tells which method to use. The order of these (and all other) parameters is not significant.

The way the form's variables are sent back to the page's owner depends on the value of the *METHOD* parameter. For *GET*, the only way to return values is to cheat: they are appended to the URL, separated by a question mark. This approach can result in URLs that are thousands of characters long. Nevertheless, this method is frequently used because it is simple.

If the *POST* method (see Fig. 7-62) is used, the body of the message contains the form's variables and their values. The & is used to separate fields; the + represents the space character. For example, the response to the widget form might be

```
customer=John+Doe&address=100+Main+St.&city=White+Plains&
state=NY&country=USA&cardno=1234567890&expires=6/98&cc=mastercard&
product=cheap&express=on
```

The string would be sent back to the server as one line, not three. If a *CHECKBOX* is not selected, it is omitted from the string. It is up to the server to make sense of this string.

Fortunately, a standard for handling forms' data is already available. It is called **CGI (Common Gateway Interface)**. Let us consider a common way of using it. Suppose that someone has an interesting database (e.g., an index of Web pages by keyword and topic) and wants to make it available to Web users. The CGI way to make the database available is to write a script (or program) that interfaces (i.e., gateways) between the database and the Web. This script is given a URL, by convention in the directory *cgi-bin*. HTTP servers know (or can be told) that when they have to invoke a method on a page located in *cgi-bin*, they are to interpret the file name as being an executable script or program and start it up.

Eventually, some user opens the form associated with our widget script and has it displayed. After the form has been filled out, the user clicks on the SUBMIT button. This action causes the browser to establish a TCP connection to the URL listed in the form's *ACTION* parameter—the script in the *cgi-bin* directory. Then the browser invokes the operation specified by the form's *METHOD*, usually *POST*. The result of this operation is that the script is started and presented (via the TCP connection, on standard input) with the long string given above. In addition, several environment variables are set. For example, the environment variable *CONTENT_LENGTH* tells how long the input string is.

At this point, most scripts need to parse their input to put it in a more convenient form. This goal can be accomplished by calling one of the many libraries or script procedures available. The script can then interact with its database in any way it wishes. For example, active maps normally use CGI scripts to take different actions depending on where the user pointed.

CGI scripts can also produce output and do many other things as well as accepting input from forms. If a hyperlink points to a CGI script, when that link is invoked, the script is started up, with several environment variables set to provide some information about the user. The script then writes a file (e.g. an HTML page) on standard output, which is shipped to the browser and interpreted there. This mechanism makes it possible for the script to generate custom Web pages on the spot.

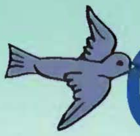
For better or worse, some Web sites that answer queries have a database of advertisements that can be selectively included in the Web page being constructed, depending on what the user is looking for. If the user is searching for "car" a General Motors ad might be displayed, whereas a search for "vacation" might produce an ad from United Airlines. These ads usually include clickable text and pictures.

7.6.4. Java

HTML makes it possible to describe how static Web pages should appear, including tables and pictures. With the cgi-bin hack, it is also possible to have a limited amount of two-way interaction (forms, etc.). However, rapid interaction with Web pages written in HTML is not possible. To make it possible to have

COMPUTER NETWORKING

THIRD EDITION



COMPUTER NETWORKS

ANDREW S. TANENBAUM

Computer Networks is the ideal introduction to today's and tomorrow's networks. This classic best-seller has been totally rewritten to reflect the networks of the late 1990s and beyond.

Author, educator, and researcher Andrew S. Tanenbaum, winner of the ACM•Karl V. Karlstrom Outstanding Educator Award, carefully explains how networks work inside, from the hardware technology up through the most popular network applications. The book takes a structured approach to networking, starting at the bottom (the physical layer) and gradually working up to the top (the application layer). The topics covered include:

- Physical layer (*e.g.*, copper, fiber, radio, and satellite communication)
- Data link layer (*e.g.*, protocol principles, HDLC, SLIP, and PPP)
- MAC Sublayer (*e.g.*, IEEE 802 LANs, bridges, new high-speed LANs)
- Network layer (*e.g.*, routing, congestion control, internetworking, IPv6)
- Transport layer (*e.g.*, transport protocol principles, TCP, network performance)
- Application layer (*e.g.*, cryptography, email, news, the Web, Java, multimedia)

In each chapter, the necessary principles are described in detail, followed by extensive examples taken from the Internet, ATM networks, and wireless networks.

Other bestselling titles by Andrew S. Tanenbaum:

Operating Systems: Design and Implementation, 2nd edition
Modern Operating Systems
Distributed Operating Systems
Structured Computer Organization, 3rd e

ISBN 0-13-349945-6

PRENTICE HALL
Upper Saddle River, NJ 07458