THIRD EDITION

COMPUTER NETWORKS

ANDREW S. TANENBAUM

Computer Networks

Third Edition

Other bestselling titles by Andrew S. Tanenbaum:

Operating Systems: Design and Implementation, 2nd edition

This now-classic text on operating systems is the only book covering both the principles of operating systems and their application to a real system. All the traditional operating systems topics are covered in detail. In addition, the principles are carefully illustrated by MINIX, a free UNIX-like operating system for personal computers. The second edition, which is expected in late 1996, will deal with the new POSIX-based MINIX 2.0 system. As with the first edition, the book will include a listing of the MINIX source code. New to the second edition is a free CD-ROM in each book containing the complete MINIX system (binary and source code).

Modern Operating Systems

This bestselling text presents the basics of both single processor and distributed computer systems. Tanenbaum covers traditional topics including processes, memory management, and files systems as well as key issues in distributed systems including the client-server model, remote procedure call, threads and distributed files servers. This practical guide uses UNIX, MS-DOS, Mach and Amoeba to illustrate operating system concepts.

Distributed Operating Systems

This text covers the fundamental concepts of distributed operating systems. Key topics include communication and synchronization, processes and processors, distributed shared memory, distributed file systems, and distributed real-time systems. The principles of distributed computing are illustrated in four detailed case studies using Mach, Amoeba, Chorus and DCE operating environments.

Structured Computer Organization, 3rd edition

This leading text looks at computer architecture as a series of levels. At the bottom is the hardware: transistors, gates, registers, adders, and other circuits. Then comes the microprogramming level. After that is the conventional machine level, with its ADD, MOVE, JUMP, and other instructions. On top of that is the operating system, which adds new facilities such as file management and virtual memory. The final chapter deals with two advanced topics: parallel computers and the design of RISC machines.

Computer Networks

Third Edition

Andrew S. Tanenbaum

Vrije Universiteit
Amsterdam, The Netherlands

Prentice Hall PTR
Upper Saddle River, New Jersey 07458

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 3 2

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*

To Suzanne, Barbara, Marvin, and Little Bram

CONTENTS

6 THE TRANSPORT LAYER 479

7 THE APPLICATION LAYER 577

PREFACE

This book is now in its third edition. Each edition has corresponded to a different phase in the way computer networks were used. When the first edition appeared in 1980, networks were an academic curiosity. When the second edition appeared in 1988, networks were used by universities and large businesses. When the third edition appeared in 1996, computer networks, especially the worldwide Internet, had become a daily reality for millions of people.

Furthermore, the networking hardware and software have completely changed since the second edition appeared. In 1988, nearly all networks were based on copper wire. Now, many are based on fiber optics or wireless communication. Proprietary networks, such as SNA, have become far less important than public networks, especially the Internet. The OSI protocols have quietly vanished, and the TCP/IP protocol suite has become dominant. In fact, so much has changed, the book has almost been rewritten from scratch.

Although Chap. 1 has the same introductory function as it did in the second edition, the contents have been completely revised and brought up to date. For example, instead of basing the book on the seven-layer OSI model, a five-layer hybrid model (shown in Fig. 1-21) is now used and introduced in Chap. 1. While not exactly identical to the TCP/IP model, it is much closer to the TCP/IP model in spirit than it is to the OSI model used in the second edition. Also, the new running examples used throughout the book—the Internet and ATM networks— are introduced here, along with some gigabit networks and other popular networks.

In Chap. 2, the focus has moved from copper wire to fiber optics and wireless communication, since these are the technologies of the future. The telephone system has become almost entirely digital in the past decade, so the material on it has been largely rewritten, with new material on broadband ISDN added. The material on cellular radio has been greatly expanded, and new material on low-orbit satellites has been added to the chapter.

The order of discussion of the data link layer and the MAC sublayer has been reversed, since experience with students shows that they understand the MAC sublayer better after they have studied the data link layer. The example protocols there have been kept, as they have proven very popular, but they have been rewritten in C. New material on the Internet and ATM data link layers has been added.

The MAC sublayer principles of Chap. 4. have been revised to reflect new protocols, including wavelength division multiplexing, wireless LANs, and digital radio. The discussion of bridges has been revised, and new material has been added on high-speed LANs.

Most of the routing algorithms of Chap. 5 have been replaced by more modern ones, including distance vector and link state routing. The sections on congestion control have been completely redone, and material on the running examples, the Internet and ATM is all new.

Chap. 6 is still about the transport layer, but here, too, major changes have occurred, primarily, the addition of a large amount of new material about the Internet, ATM, and network performance.

Chap. 7, on the application layer, is now the longest chapter in the book. The material on network security has been doubled in length, and new material has been added on DNS, SNMP, email, USENET, the World Wide Web, HTML, Java, multimedia, video on demand, and the MBone.

Of the 395 figures in the third edition, 276 (70 percent) are completely new and some of the others have been revised. Of the 371 references to the literature, 282 (76 percent) are to books and papers that have appeared since the second edition was published. Of these, over 100 are to works published in 1995 and 1996 alone. All in all, probably 75 percent of the entire book is brand new, and parts of the remaining 25 percent have been heavily revised. Since this is effectively a new book, the cover was redesigned to avoid confusion with the second edition.

Computer books are full of acronyms. This one is no exception. By the time you are finished reading this one, all of the following should ring a bell: AAL, AMPS, ARP, ASN, ATM, BGP, CDMA, CDPD, CSMA, DQDB, DNS, FAQ, FDM, FTP, FTTC, FTTH, GSM, HDLC, HEC, HIPPI, IAB, ICMP, IDEA, IETF, IPv6, ISO, ITU, LATA, MAC, MACA, MAN, MIB, MIME, NAP, NNTP, NSA, NSAP, OSI, OSPF, PCM, PCN, PCS, PEM, PGP, PPP, PSTN, PTT, PVC, QAM, RARP, RFC, RSA, SABME, SAP, SAR, SDH, SDLC, SHA, SMI, SNA, SNMP, SNRME, SPX, TCP, UDP, VHF, VLF, VSAT, WARC, WDM, WWV, and WWW. But don't worry. Each one will be carefully defined before it is used.

To help instructors using this book as a text for course, the author has prepared three teaching aids:

- A problem solutions manual.
- PostScript files containing all the figures (for making overhead sheets).
- A simulator (written in C) for the example protocols of Chap. 3.

The solutions manual is available from Prentice Hall (but only to instructors). The file with the figures and the simulator are available via the World Wide Web. To get them, please see the author's home page: *http://www.cs.vu.nl/~ast/*.

The book was typeset in Times Roman using Troff, which, after all these years, is still the only way to go. While Troff is not as trendy as WYSIWYG systems, the reader is invited to compare the typesetting quality of this book with books produced by WYSIWYG systems. My only concession to PCs and desktop publishing is that for the first time, the art was produced using Adobe Illustrator, instead of being drawn on paper. Also for the first time, the book was produced entirely electronically. The PostScript output from Troff was sent over the Internet to the printer, where the film for making the offset plates was produced. No intermediate paper copy was printed and photographed, as is normally done.

Many people helped me during the course of the third edition. I would especially like to thank Chase Bailey, Saniya Ben Hassen, Nathaniel Borenstein, Ron Cocchi, Dave Crocker, Wiebren de Jonge, Carl Ellison, M. Rasit Eskicioglu, John Evans, Mario Gerla, Mike Goguen, Paul Green, Dick Grune, Wayne Hathaway, Franz Hauck, Jack Holtzman, Gerard Holzmann, Philip Homburg, Peter Honeyman, Raj Jain, Dave Johnson, Charlie Kaufman, Vinay Kumar, Jorg Liebeherr, Paul Mockapetris, Carol Orange, Craig Partridge, Charlie Perkins, Thomas Powell, Greg Sharp, Anne Steegstra, George Swallow, Mark Taylor, Peter van der Linden, Hans van Staveren, Maarten van Steen, Kees Verstoep, Stephen Walters, Michael Weintraub, Joseph Wilkes, and Stephen Wolff. Special thanks go to Radia Perlman for many helpful suggestions. My students have also helped in many ways. I would like to single out Martijn Bot, Wilbert de Graaf, Flavio del Pomo, and Arnold de Wit for their assistance.

My editor at Prentice Hall, Mary Franz, provided me with more reading material than I had consumed in the previous 10 years. She was also helpful in numerous other ways, small, medium, large, and jumbo. My production editor, Camille Trentacoste, taught me about people of snow, 8-up flats, fax [sic], and other important items, while performing yeoperson's service with a Picky Author and a tight schedule.

Finally, we come to the most important people. Suzanne, Barbara, Marvin, and even little Bram, have been through this routine before. They endure it with infinite patience and good grace. Thank you.

ANDREW S. TANENBAUM

1

INTRODUCTION

Each of the past three centuries has been dominated by a single technology. The 18th Century was the time of the great mechanical systems accompanying the Industrial Revolution. The 19th Century was the age of the steam engine. During the 20th Century, the key technology has been information gathering, processing, and distribution. Among other developments, we have seen the installation of worldwide telephone networks, the invention of radio and television, the birth and unprecedented growth of the computer industry, and the launching of communication satellites.

Due to rapid technological progress, these areas are rapidly converging, and the differences between collecting, transporting, storing, and processing information are quickly disappearing. Organizations with hundreds of offices spread over a wide geographical area routinely expect to be able to examine the current status of even their most remote outpost at the push of a button. As our ability to gather, process, and distribute information grows, the demand for even more sophisticated information processing grows even faster.

Although the computer industry is young compared to other industries (e.g., automobiles and air transportation), computers have made spectacular progress in a short time. During the first two decades of their existence, computer systems were highly centralized, usually within a single large room. Not infrequently, this room had glass walls, through which visitors could gawk at the great electronic wonder inside. A medium-size company or university might have had one or two 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.

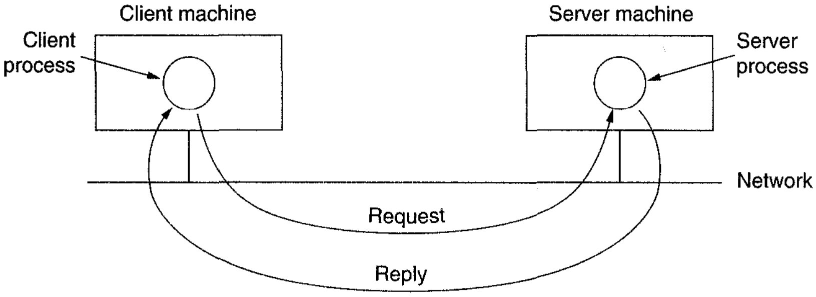


Fig. 1-1. The client-server model.

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 may actually be received (heard) by many people, only Watson responds. The others just ignore it. Another example is an airport announcement asking all flight 644 passengers to report to gate 12.

Broadcast systems generally also allow the possibility of addressing a packet to *all* destinations by using a special code in the address field. When a packet with this code is transmitted, it is received and processed by every machine on the network. This mode of operation is called **broadcasting**. Some broadcast systems also support transmission to a subset of the machines, something known as **multicasting**. One possible scheme is to reserve one bit to indicate multicasting. The remaining $n - 1$ address bits can hold a group number. Each machine can "subscribe" to any or all of the groups. When a packet is sent to a certain group, it is delivered to all machines subscribing to that group.

In contrast, **point-to-point** networks consist of many connections between individual pairs of machines. To go from the source to the destination, a packet on this type of network may have to first visit one or more intermediate machines. Often multiple routes, of different lengths are possible, so routing algorithms play an important role in point-to-point networks. As a general rule (although there are many exceptions), smaller, geographically localized networks tend to use broadcasting, whereas larger networks usually are point-to-point.

Interprocessor distance	Processors located in same	Example
0.1 m	Circuit board	Data flow machine
1 m	System	Multicomputer
10 m	Room	Local area network
100 m	Building	
1 km	Campus	
10 km	City	Metropolitan area network
100 km	Country	Wide area network
1,000 km	Continent	
10,000 km	Planet	The internet

Fig. 1-2. Classification of interconnected processors by scale.

An alternative criterion for classifying networks is their scale. In Fig. 1-2 we give a classification of multiple processor systems arranged by their physical size. At the top are **data flow machines**, highly parallel computers with many functional units all working on the same program. Next come the **multicomputers**, systems that communicate by sending messages over very short, very fast buses. Beyond the multicomputers are the true networks, computers that communicate by exchanging messages over longer cables. These can be divided into local, metropolitan, and wide area networks. Finally, the connection of two or more networks is called an internetwork. The worldwide Internet is a well-known example of an internetwork. Distance is important as a classification metric because different techniques are used at different scales. In this book we will be concerned with only the true networks and their interconnection. Below we give a brief introduction to the subject of network hardware.

1.2.1. Local Area Networks

Local area networks, generally called **LANs**, are privately-owned networks within a single building or campus of up to a few kilometers in size. They are widely used to connect personal computers and workstations in company offices and factories to share resources (e.g., printers) and exchange information. LANs are distinguished from other kinds of networks by three characteristics: (1) their size, (2) their transmission technology, and (3) their topology.

LANs are restricted in size, which means that the worst-case transmission time is bounded and known in advance. Knowing this bound makes it possible to use certain kinds of designs that would not otherwise be possible. It also simplifies network management.

LANs often use a transmission technology consisting of a single cable to which all the machines are attached, like the telephone company party lines once used in rural areas. Traditional LANs run at speeds of 10 to 100 Mbps, have low delay (tens of microseconds), and make very few errors. Newer LANs may operate at higher speeds, up to hundreds of megabits/sec. In this book, we will adhere to tradition and measure line speeds in megabits/sec (Mbps), not megabytes/sec (MB/sec). A megabit is 1,000,000 bits, not 1,048,576 (2^{20}) bits.

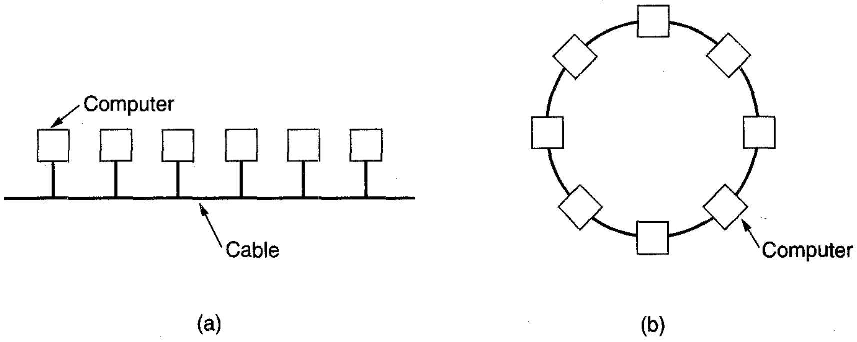


Fig. 1-3. Two broadcast networks. (a) Bus. (b) Ring.

Various topologies are possible for broadcast LANs. Figure 1-3 shows two of them. In a bus (i.e., a linear cable) network, at any instant one machine is the master and is allowed to transmit. All other machines are required to refrain from sending. An arbitration mechanism is needed to resolve conflicts when two or more machines want to transmit simultaneously. The arbitration mechanism may be centralized or distributed. IEEE 802.3, popularly called **Ethernet**$^{\text{TM}}$, for example, is a bus-based broadcast network with decentralized control operating at 10 or 100 Mbps. Computers on an Ethernet can transmit whenever they want to; if two or more packets collide, each computer just waits a random time and tries again later.

A second type of broadcast system is the ring. In a ring, each bit propagates around on its own, not waiting for the rest of the packet to which it belongs. Typically, each bit circumnavigates the entire ring in the time it takes to transmit a few bits, often before the complete packet has even been transmitted. Like all other broadcast systems, some rule is needed for arbitrating simultaneous accesses to the ring. Various methods are in use and will be discussed later in this book. IEEE 802.5 (the IBM token ring), is a popular ring-based LAN operating at 4 and 16 Mbps.

Broadcast networks can be further divided into static and dynamic, depending on how the channel is allocated. A typical static allocation would be to divide up time into discrete intervals and run a round robin algorithm, allowing each machine to broadcast only when its time slot comes up. Static allocation wastes channel capacity when a machine has nothing to say during its allocated slot, so most systems attempt to allocate the channel dynamically (i.e., on demand).

Dynamic allocation methods for a common channel are either centralized or decentralized. In the centralized channel allocation method, there is a single entity, for example a bus arbitration unit, which determines who goes next. It might do this by accepting requests and making a decision according to some internal algorithm. In the decentralized channel allocation method, there is no central entity; each machine must decide for itself whether or not to transmit. You might think that this always leads to chaos, but it does not. Later we will study many algorithms designed to bring order out of the potential chaos.

The other kind of LAN is built using point-to-point lines. Individual lines connect a specific machine with another specific machine. Such a LAN is really a miniature wide area network. We will look at these later.

1.2.2. Metropolitan Area Networks

A **metropolitan area network**, or **MAN** (plural: MANs, not MEN) is basically a bigger version of a LAN and normally uses similar technology. It might cover a group of nearby corporate offices or a city and might be either private or public. A MAN can support both data and voice, and might even be related to the local cable television network. A MAN just has one or two cables and does not contain switching elements, which shunt packets over one of several potential output lines. Not having to switch simplifies the design.

The main reason for even distinguishing MANs as a special category is that a standard has been adopted for them, and this standard is now being implemented. It is called **DQDB** (**Distributed Queue Dual Bus**) or for people who prefer numbers to letters, 802.6 (the number of the IEEE standard that defines it). DQDB consists of two unidirectional buses (cables) to which all the computers are connected, as shown in Fig. 1-4. Each bus has a head-end, a device that initiates transmission activity. Traffic that is destined for a computer to the right of the sender uses the upper bus. Traffic to the left uses the lower one.

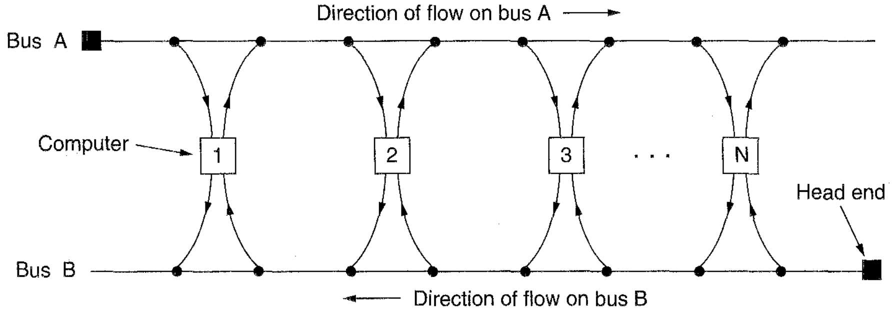


Fig. 1-4. Architecture of the DQDB metropolitan area network.

A key aspect of a MAN is that there is a broadcast medium (for 802.6, two cables) to which all the computers are attached. This greatly simplifies the design compared to other kinds of networks. We will discuss DQDB in more detail in Chap. 4.

1.2.3. Wide Area Networks

A **wide area network**, or **WAN**, spans a large geographical area, often a country or continent. It contains a collection of machines intended for running user (i.e., application) programs. We will follow traditional usage and call these machines **hosts**. The term **end system** is sometimes also used in the literature. The hosts are connected by a **communication subnet**, or just **subnet** for short. The job of the subnet is to carry messages from host to host, just as the telephone system carries words from speaker to listener. By separating the pure communication aspects of the network (the subnet) from the application aspects (the hosts), the complete network design is greatly simplified.

In most wide area networks, the subnet consists of two distinct components: transmission lines and switching elements. Transmission lines (also called **circuits**, **channels**, or **trunks**) move bits between machines.

The switching elements are specialized computers used to connect two or more transmission lines. When data arrive on an incoming line, the switching element must choose an outgoing line to forward them on. Unfortunately, there is no standard terminology used to name these computers. They are variously called **packet switching nodes**, **intermediate systems**, and **data switching exchanges**, among other things. As a generic term for the switching computers, we will use the word **router**, but the reader should be aware that no consensus on terminology exists here. In this model, shown in Fig. 1-5, each host is generally connected to a LAN on which a router is present, although in some cases a host can be connected directly to a router. The collection of communication lines and routers (but not the hosts) form the subnet.

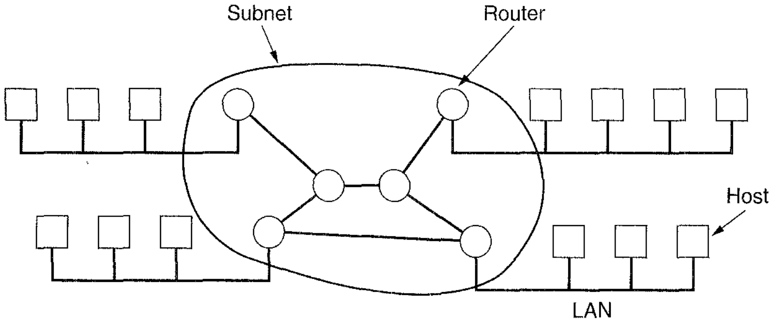


Fig. 1-5. Relation between hosts and the subnet.

An aside about the term "subnet" is worth making. Originally, its only meaning was the collection of routers and communication lines that moved packets from the source host to the destination host. However, some years later, it also acquired a second meaning in conjunction with network addressing (which we will discuss in Chap. 5). Hence the term has a certain ambiguity about it. Unfortunately, no widely-used alternative exists for its initial meaning, so with some hesitation we will use it in both senses. From the context, it will always be clear which is meant.

In most WANs, the network contains numerous cables or telephone lines, each one connecting a pair of routers. If two routers that do not share a cable nevertheless wish to communicate, they must do this indirectly, via other routers. When a packet is sent from one router to another via one or more intermediate routers, the packet is received at each intermediate router in its entirety, stored there until the required output line is free, and then forwarded. A subnet using this principle is called a **point-to-point**, **store-and-forward**, or **packet-switched** subnet. Nearly all wide area networks (except those using satellites) have store-and-forward subnets. When the packets are small and all the same size, they are often called **cells**.

When a point-to-point subnet is used, an important design issue is what the router interconnection topology should look like. Figure 1-6 shows several possible topologies. Local networks that were designed as such usually have a symmetric topology. In contrast, wide area networks typically have irregular topologies.

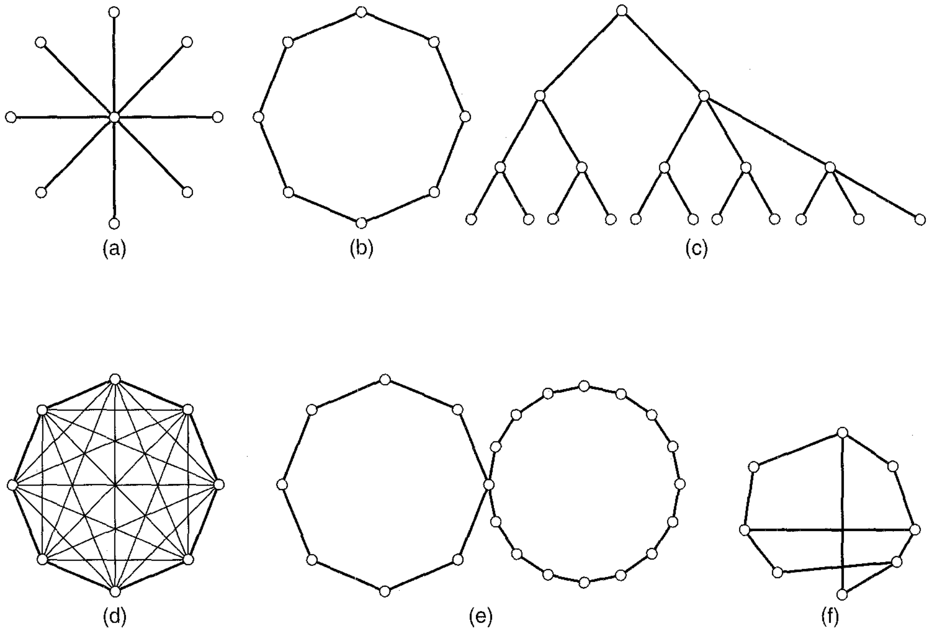


Fig. 1-6. Some possible topologies for a point-to-point subnet. (a) Star. (b) Ring. (c) Tree. (d) Complete. (e) Intersecting rings. (f) Irregular.

A second possibility for a WAN is a satellite or ground radio system. Each router has an antenna through which it can send and receive. All routers can hear the output *from* the satellite, and in some cases they can also hear the upward transmissions of their fellow routers *to* the satellite as well. Sometimes the routers are connected to a substantial point-to-point subnet, with only some of them having a satellite antenna. Satellite networks are inherently broadcast and are most useful when the broadcast property is important.

1.2.4. Wireless Networks

Mobile computers, such as notebook computers and personal digital assistants (PDAs), are the fastest-growing segment of the computer industry. Many of the owners of these computers have desktop machines on LANs and WANs back at the office and want to be connected to their home base even when away from home or en route. Since having a wired connection is impossible in cars and airplanes, there is a lot of interest in wireless networks. In this section we will briefly introduce this topic. (Note: by section, we mean those portions of the book with a three-part number such as 1.2.4.)

Actually, digital wireless communication is not a new idea. As early as 1901, the Italian physicist Guglielmo Marconi demonstrated a ship-to-shore wireless telegraph using Morse Code (dots and dashes are binary, after all). Modern digital wireless systems have better performance, but the basic idea is the same. Additional information about these systems can be found in (Garg and Wilkes, 1996; and Pahlavan et al., 1995).

Wireless networks have many uses. A common one is the portable office. People on the road often want to use their portable electronic equipment to send and receive telephone calls, faxes, and electronic mail, read remote files, login on remote machines, and so on, and do this from anywhere on land, sea, or air.

Wireless networks are of great value to fleets of trucks, taxis, buses, and repairpersons for keeping in contact with home. Another use is for rescue workers at disaster sites (fires, floods, earthquakes, etc.) where the telephone system has been destroyed. Computers there can send messages, keep records, and so on.

Finally, wireless networks are important to the military. If you have to be able to fight a war anywhere on earth on short notice, counting on using the local networking infrastructure is probably not a good idea. It is better to bring your own.

Although wireless networking and mobile computing are often related, they are not identical, as Fig. 1-7 shows. Portable computers are sometimes wired. For example, if a traveler plugs a portable computer into the telephone jack in a hotel, we have mobility without a wireless network. Another example is someone carrying a portable computer along as he inspects a train for technical problems. Here a long cord can trail along behind (vacuum cleaner model).

Wireless	Mobile	Applications
No	No	Stationary workstations in offices
No	Yes	Using a portable in a hotel; train maintenance
Yes	No	LANs in older, unwired buildings
Yes	Yes	Portable office; PDA for store inventory

Fig. 1-7. Combinations of wireless networks and mobile computing.

On the other hand, some wireless computers are not portable. An important example here is a company that owns an older building that does not have network cabling installed and wants to connect its computers. Installing a wireless LAN may require little more than buying a small box with some electronics and setting up some antennas. This solution may be cheaper than wiring the building.

Although wireless LANs are easy to install, they also have some disadvantages. Typically they have a capacity of 1–2 Mbps, which is much slower than wired LANs. The error rates are often much higher, too, and the transmissions from different computers can interfere with one another.

But of course, there are also the true mobile, wireless applications, ranging from the portable office to people walking around a store with a PDA doing inventory. At many busy airports, car rental return clerks work out in the parking lot with wireless portable computers. They type in the license plate number of returning cars, and their portable, which has a built-in printer, calls the main computer, gets the rental information, and prints out the bill on the spot. True mobile computing is discussed further in (Forman and Zahorjan, 1994).

Wireless networks come in many forms. Some universities are already installing antennas all over campus to allow students to sit under the trees and consult the library's card catalog. Here the computers communicate directly with the wireless LAN in digital form. Another possibility is using a cellular (i.e., portable) telephone with a traditional analog modem. Direct digital cellular service, called **CDPD** (**Cellular Digital Packet Data**) is becoming available in many cities. We will study it in Chap. 4.

Finally, it is possible to have different combinations of wired and wireless networking. For example, in Fig. 1-8(a), we depict an airplane with a number of people using modems and seat-back telephones to call the office. Each call is independent of the other ones. A much more efficient option, however, is the flying LAN of Fig. 1-8(b). Here each seat comes equipped with an Ethernet connector into which passengers can plug their computers. A single router on the aircraft maintains a radio link with some router on the ground, changing routers as it flies along. This configuration is just a traditional LAN, except that its connection to the outside world happens to be a radio link instead of a hardwired line.

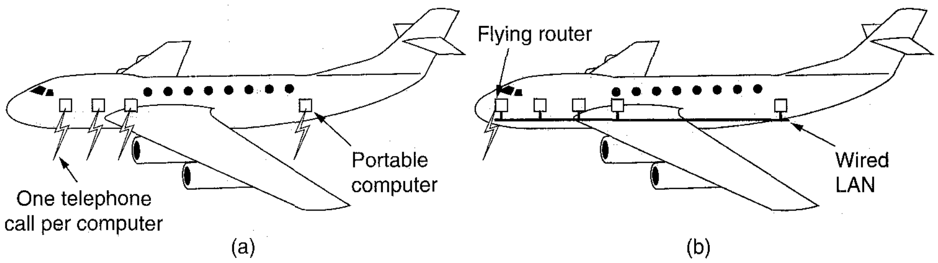


Fig. 1-8. (a) Individual mobile computers. (b) A flying LAN.

While many people believe that wireless portable computers are the wave of the future, at least one dissenting voice has been heard. Bob Metcalfe, the inventor of Ethernet, has written: "Mobile wireless computers are like mobile pipeless bathrooms—portapotties. They will be common on vehicles, and at construction sites, and rock concerts. My advice is to wire up your home and stay there" (Metcalfe, 1995). Will most people follow Metcalfe's advice? Time will tell.

1.2.5. Internetworks

Many networks exist in the world, often with different hardware and software. People connected to one network often want to communicate with people attached to a different one. This desire requires connecting together different, and frequently incompatible networks, sometimes by using machines called **gateways** to make the connection and provide the necessary translation, both in terms of hardware and software. A collection of interconnected networks is called an **internetwork** or just **internet**.

A common form of internet is a collection of LANs connected by a WAN. In fact, if we were to replace the label "subnet" in Fig. 1-5 by "WAN," nothing else in the figure would have to change. The only real distinction between a subnet and a WAN in this case is whether or not hosts are present. If the system within the closed curve contains only routers, it is a subnet. If it contains both routers and hosts with their own users, it is a WAN.

To avoid confusion, please note that the word "internet" will always be used in this book in a generic sense. In contrast, the **Internet** (note uppercase I) means a specific worldwide internet that is widely used to connect universities, government offices, companies, and of late, private individuals. We will have much to say about both internets and the Internet later in this book.

Subnets, networks, and internetworks are often confused. Subnet makes the most sense in the context of a wide area network, where it refers to the collection of routers and communication lines owned by the network operator, for example, companies like America Online and CompuServe. As an analogy, the telephone system consists of telephone switching offices connected to each other by high-speed lines, and to houses and businesses by low-speed lines. These lines and equipment, owned and managed by the telephone company, form the subnet of the telephone system. The telephones themselves (the hosts in this analogy) are not part of the subnet. The combination of a subnet and its hosts forms a network. In the case of a LAN, the cable and the hosts form the network. There really is no subnet.

An internetwork is formed when distinct networks are connected together. In our view, connecting a LAN and a WAN or connecting two LANs forms an internetwork, but there is little agreement in the industry over terminology in this area.

1.3. NETWORK SOFTWARE

The first computer networks were designed with the hardware as the main concern and the software as an afterthought. This strategy no longer works. Network software is now highly structured. In the following sections we examine the software structuring technique in some detail. The method described here forms the keystone of the entire book and will occur repeatedly later on.

1.3.1. Protocol Hierarchies

To reduce their design complexity, most networks are organized as a series of **layers** or **levels**, each one built upon the one below it. The number of layers, the name of each layer, the contents of each layer, and the function of each layer differ from network to network. However, in all networks, the purpose of each layer is to offer certain services to the higher layers, shielding those layers from the details of how the offered services are actually implemented.

Layer *n* on one machine carries on a conversation with layer *n* on another machine. The rules and conventions used in this conversation are collectively known as the layer *n* **protocol**. Basically, a protocol is an agreement between the communicating parties on how communication is to proceed. As an analogy, when a woman is introduced to a man, she may choose to stick out her hand. He, in turn, may decide either to shake it or kiss it, depending, for example, on whether she is an American lawyer at a business meeting or a European princess at a formal ball. Violating the protocol will make communication more difficult, if not impossible.

A five-layer network is illustrated in Fig. 1-9. The entities comprising the corresponding layers on different machines are called **peers**. In other words, it is the peers that communicate using the protocol.

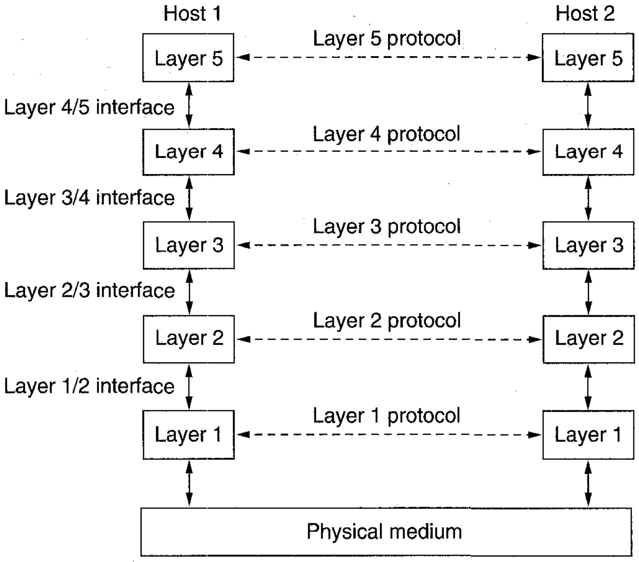


Fig. 1-9. Layers, protocols, and interfaces.

In reality, no data are directly transferred from layer *n* on one machine to layer *n* on another machine. Instead, each layer passes data and control information to the layer immediately below it, until the lowest layer is reached. Below layer 1 is the **physical medium** through which actual communication occurs. In Fig. 1-9, virtual communication is shown by dotted lines and physical communication by solid lines.

Between each pair of adjacent layers there is an **interface**. The interface defines which primitive operations and services the lower layer offers to the upper one. When network designers decide how many layers to include in a network and what each one should do, one of the most important considerations is defining clean interfaces between the layers. Doing so, in turn, requires that each layer perform a specific collection of well-understood functions. In addition to minimizing the amount of information that must be passed between layers, clean-cut interfaces also make it simpler to replace the implementation of one layer with a completely different implementation (e.g., all the telephone lines are replaced by satellite channels), because all that is required of the new implementation is that it offers exactly the same set of services to its upstairs neighbor as the old implementation did.

A set of layers and protocols is called a **network architecture**. The specification of an architecture must contain enough information to allow an implementer to write the program or build the hardware for each layer so that it will correctly obey the appropriate protocol. Neither the details of the implementation nor the specification of the interfaces are part of the architecture because these are hidden away inside the machines and not visible from the outside. It is not even necessary that the interfaces on all machines in a network be the same, provided that each machine can correctly use all the protocols. A list of protocols used by a certain system, one protocol per layer, is called a **protocol stack**. The subjects of network architectures, protocol stacks, and the protocols themselves are the principal topics of this book.

An analogy may help explain the idea of multilayer communication. Imagine two philosophers (peer processes in layer 3), one of whom speaks Urdu and English and one of whom speaks Chinese and French. Since they have no common language, they each engage a translator (peer processes at layer 2), each of whom in turn contacts a secretary (peer processes in layer 1). Philosopher 1 wishes to convey his affection for *oryctolagus cuniculus* to his peer. To do so, he passes a message (in English) across the 2/3 interface, to his translator, saying "I like rabbits," as illustrated in Fig. 1-10. The translators have agreed on a neutral language, Dutch, so the message is converted to "Ik hou van konijnen." The choice of language is the layer 2 protocol and is up to the layer 2 peer processes.

The translator then gives the message to a secretary for transmission, by, for example, fax (the layer 1 protocol). When the message arrives, it is translated into French and passed across the 2/3 interface to philosopher 2. Note that each protocol is completely independent of the other ones as long as the interfaces are not changed. The translators can switch from Dutch to say, Finnish, at will, provided that they both agree, and neither changes his interface with either layer 1 or

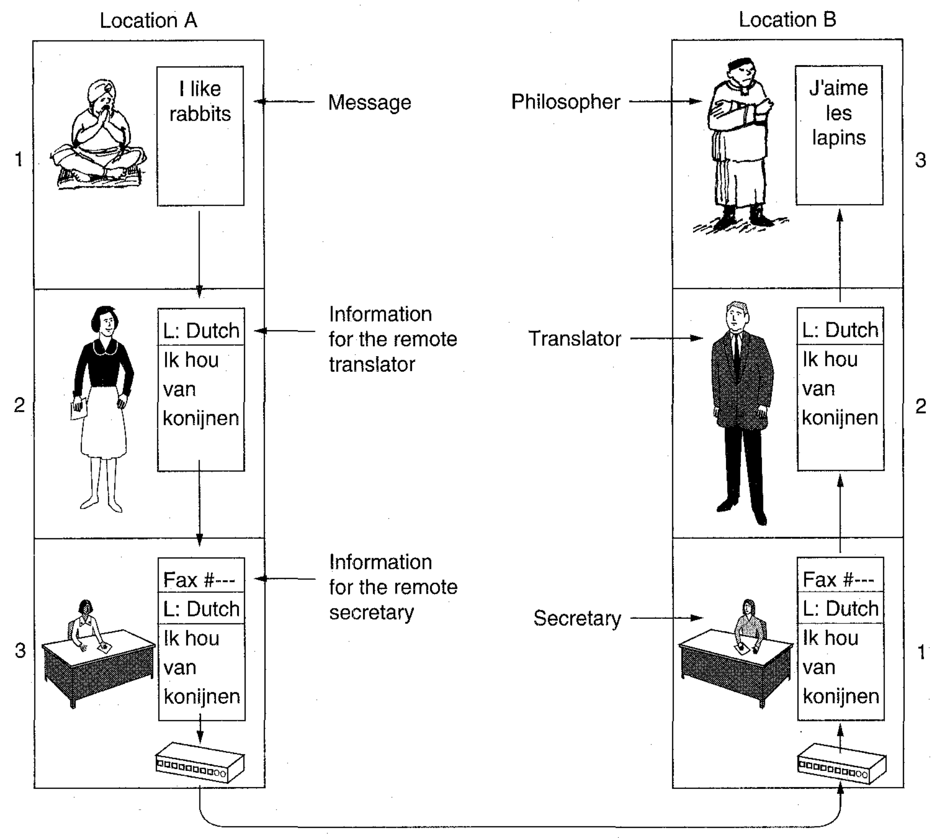


Fig. 1-10. The philosopher-translator-secretary architecture.

layer 3. Similarly the secretaries can switch from fax to email, or telephone without disturbing (or even informing) the other layers. Each process may add some information intended only for its peer. This information is not passed upward to the layer above.

Now consider a more technical example: how to provide communication to the top layer of the five-layer network in Fig. 1-11. A message, *M*, is produced by an application process running in layer 5 and given to layer 4 for transmission. Layer 4 puts a **header** in front of the message to identify the message and passes the result to layer 3. The header includes control information, such as sequence numbers, to allow layer 4 on the destination machine to deliver messages in the right order if the lower layers do not maintain sequence. In some layers, headers also contain sizes, times, and other control fields.

In many networks, there is no limit to the size of messages transmitted in the layer 4 protocol, but there is nearly always a limit imposed by the layer 3 protocol. Consequently, layer 3 must break up the incoming messages into smaller

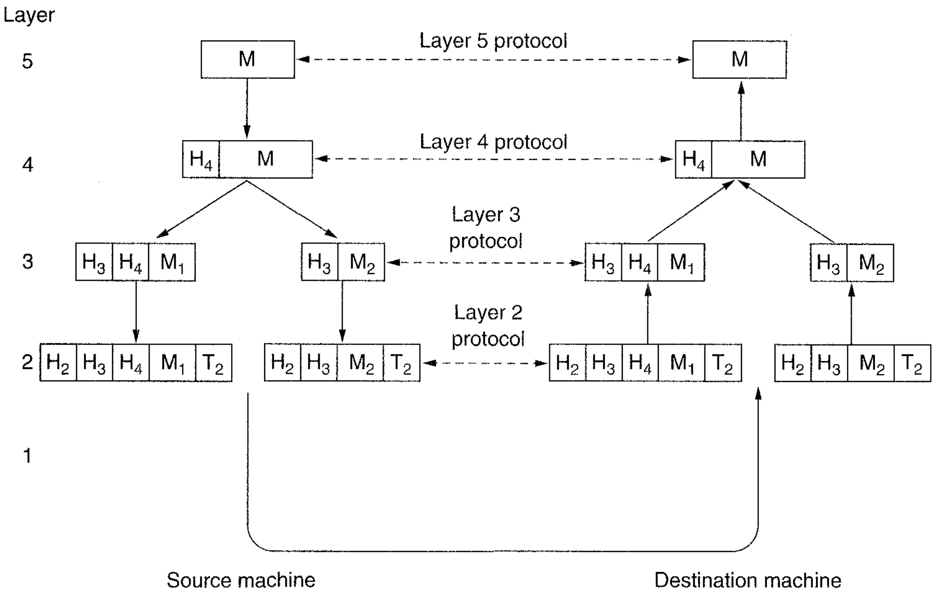


Fig. 1-11. Example information flow supporting virtual communication in layer 5.

units, packets, prepending a layer 3 header to each packet. In this example, M is split into two parts, M_1 and M_2.

Layer 3 decides which of the outgoing lines to use and passes the packets to layer 2. Layer 2 adds not only a header to each piece, but also a trailer, and gives the resulting unit to layer 1 for physical transmission. At the receiving machine the message moves upward, from layer to layer, with headers being stripped off as it progresses. None of the headers for layers below n are passed up to layer n.

The important thing to understand about Fig. 1-11 is the relation between the virtual and actual communication and the difference between protocols and interfaces. The peer processes in layer 4, for example, conceptually think of their communication as being "horizontal," using the layer 4 protocol. Each one is likely to have a procedure called something like *SendToOtherSide* and *GetFromOtherSide*, even though these procedures actually communicate with lower layers across the 3/4 interface, not with the other side.

The peer process abstraction is crucial to all network design. Using it, the unmanageable task of designing the complete network can be broken into several smaller, manageable, design problems, namely the design of the individual layers.

Although Section 1-3 is called "Network Software," it is worth pointing out that the lower layers of a protocol hierarchy are frequently implemented in hardware or firmware. Nevertheless, complex protocol algorithms are involved, even if they are embedded (in whole or in part) in hardware.

1.3.2. Design Issues for the Layers

Some of the key design issues that occur in computer networking are present in several layers. Below, we will briefly mention some of the more important ones.

Every layer needs a mechanism for identifying senders and receivers. Since a network normally has many computers, some of which have multiple processes, a means is needed for a process on one machine to specify with whom it wants to talk. As a consequence of having multiple destinations, some form of addressing is needed in order to specify a specific destination.

Another set of design decisions concerns the rules for data transfer. In some systems, data only travel in one direction (**simplex communication**). In others they can travel in either direction, but not simultaneously (**half-duplex communication**). In still others they travel in both directions at once (**full-duplex communication**). The protocol must also determine how many logical channels the connection corresponds to, and what their priorities are. Many networks provide at least two logical channels per connection, one for normal data and one for urgent data.

Error control is an important issue because physical communication circuits are not perfect. Many error-detecting and error-correcting codes are known, but both ends of the connection must agree on which one is being used. In addition, the receiver must have some way of telling the sender which messages have been correctly received and which have not.

Not all communication channels preserve the order of messages sent on them. To deal with a possible loss of sequencing, the protocol must make explicit provision for the receiver to allow the pieces to be put back together properly. An obvious solution is to number the pieces, but this solution still leaves open the question of what should be done with pieces that arrive out of order.

An issue that occurs at every level is how to keep a fast sender from swamping a slow receiver with data. Various solutions have been proposed and will be discussed later. Some of them involve some kind of feedback from the receiver to the sender, either directly or indirectly, about the receiver's current situation. Others limit the sender to an agreed upon transmission rate.

Another problem that must be solved at several levels is the inability of all processes to accept arbitrarily long messages. This property leads to mechanisms for disassembling, transmitting, and then reassembling messages. A related issue is what to do when processes insist upon transmitting data in units that are so small that sending each one separately is inefficient. Here the solution is to gather together several small messages heading toward a common destination into a single large message and dismember the large message at the other side.

When it is inconvenient or expensive to set up a separate connection for each pair of communicating processes, the underlying layer may decide to use the same connection for multiple, unrelated conversations. As long as this multiplexing and demultiplexing is done transparently, it can be used by any layer. Multiplexing is needed in the physical layer, for example, where all the traffic for all connections has to be sent over at most a few physical circuits.

When there are multiple paths between source and destination, a route must be chosen. Sometimes this decision must be split over two or more layers. For example, to send data from London to Rome, a high-level decision might have to be made to go via France or Germany based on their respective privacy laws, and a low-level decision might have to be made to choose one of the many available circuits based on the current traffic load.

1.3.3. Interfaces and Services

The function of each layer is to provide services to the layer above it. In this section we will look at precisely what a service is in more detail, but first we will give some terminology.

The active elements in each layer are often called **entities**. An entity can be a software entity (such as a process), or a hardware entity (such as an intelligent I/O chip). Entities in the same layer on different machines are called **peer entities**. The entities in layer n implement a service used by layer $n + 1$. In this case layer n is called the **service provider** and layer $n + 1$ is called the **service user**. Layer n may use the services of layer $n - 1$ in order to provide its service. It may offer several classes of service, for example, fast, expensive communication and slow, cheap communication.

Services are available at **SAP**s (**Service Access Points**), The layer n SAPs are the places where layer $n + 1$ can access the services offered. Each SAP has an address that uniquely identifies it. To make this point clearer, the SAPs in the telephone system are the sockets into which modular telephones can be plugged, and the SAP addresses are the telephone numbers of these sockets. To call someone, you must know the callee's SAP address. Similarly, in the postal system, the SAP addresses are street addresses and post office box numbers. To send a letter, you must know the addressee's SAP address.

In order for two layers to exchange information, there has to be an agreed upon set of rules about the interface. At a typical interface, the layer $n + 1$ entity passes an **IDU** (**Interface Data Unit**) to the layer n entity through the SAP as shown in Fig. 1-12. The IDU consists of an **SDU** (**Service Data Unit**) and some control information. The SDU is the information passed across the network to the peer entity and then up to layer $n + 1$. The control information is needed to help the lower layer do its job (e.g., the number of bytes in the SDU) but is not part of the data itself.

In order to transfer the SDU, the layer n entity may have to fragment it into several pieces, each of which is given a header and sent as a separate **PDU** (**Protocol Data Unit**) such as a packet. The PDU headers are used by the peer entities

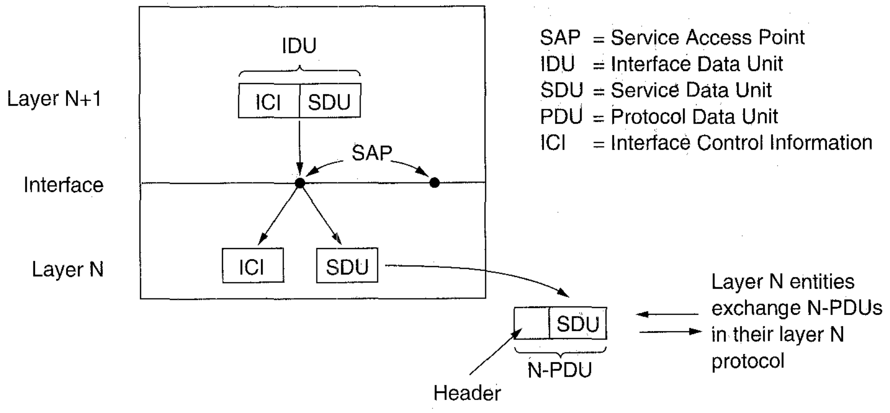


Fig. 1-12. Relation between layers at an interface.

to carry out their peer protocol. They identify which PDUs contain data and which contain control information, provide sequence numbers and counts, and so on.

1.3.4. Connection-Oriented and Connectionless Services

Layers can offer two different types of service to the layers above them: connection-oriented and connectionless. In this section we will look at these two types and examine the differences between them.

Connection-oriented service is modeled after the telephone system. To talk to someone, you pick up the phone, dial the number, talk, and then hang up. Similarly, to use a connection-oriented network service, the service user first establishes a connection, uses the connection, and then releases the connection. The essential aspect of a connection is that it acts like a tube: the sender pushes objects (bits) in at one end, and the receiver takes them out in the same order at the other end.

In contrast, **connectionless service** is modeled after the postal system. Each message (letter) carries the full destination address, and each one is routed through the system independent of all the others. Normally, when two messages are sent to the same destination, the first one sent will be the first one to arrive. However, it is possible that the first one sent can be delayed so that the second one arrives first. With a connection-oriented service this is impossible.

Each service can be characterized by a **quality of service**. Some services are reliable in the sense that they never lose data. Usually, a reliable service is implemented by having the receiver acknowledge the receipt of each message, so the sender is sure that it arrived. The acknowledgement process introduces overhead and delays, which are often worth it but are sometimes undesirable.

A typical situation in which a reliable connection-oriented service is appropriate is file transfer. The owner of the file wants to be sure that all the bits arrive correctly and in the same order they were sent. Very few file transfer customers would prefer a service that occasionally scrambles or loses a few bits, even if it is much faster.

Reliable connection-oriented service has two minor variations: message sequences and byte streams. In the former, the message boundaries are preserved. When two 1-KB messages are sent, they arrive as two distinct 1-KB messages, never as one 2-KB message. (Note: KB means kilobytes; kb means kilobits.) In the latter, the connection is simply a stream of bytes, with no message boundaries. When 2K bytes arrive at the receiver, there is no way to tell if they were sent as one 2-KB message, two 1-KB messages, or 2048 1-byte messages. If the pages of a book are sent over a network to a phototypesetter as separate messages, it might be important to preserve the message boundaries. On the other hand, with a terminal logging into a remote timesharing system, a byte stream from the terminal to the computer is all that is needed.

As mentioned above, for some applications, the delays introduced by acknowledgements are unacceptable. One such application is digitized voice traffic. It is preferable for telephone users to hear a bit of noise on the line or a garbled word from time to time than to introduce a delay to wait for acknowledgements. Similarly, when transmitting a video film, having a few pixels wrong is no problem, but having the film jerk along as the flow stops to correct errors is very irritating.

Not all applications require connections. For example, as electronic mail becomes more common, can electronic junk mail be far behind? The electronic junk mail sender probably does not want to go to the trouble of setting up and later tearing down a connection just to send one item. Nor is 100 percent reliable delivery essential, especially if it costs more. All that is needed is a way to send a single message that has a high probability of arrival, but no guarantee. Unreliable (meaning not acknowledged) connectionless service is often called **datagram service**, in analogy with telegram service, which also does not provide an acknowledgement back to the sender.

In other situations, the convenience of not having to establish a connection to send one short message is desired, but reliability is essential. The **acknowledged datagram service** can be provided for these applications. It is like sending a registered letter and requesting a return receipt. When the receipt comes back, the sender is absolutely sure that the letter was delivered to the intended party and not lost along the way.

Still another service is the **request-reply service**. In this service the sender transmits a single datagram containing a request; the reply contains the answer. For example, a query to the local library asking where Uighur is spoken falls into this category. Request-reply is commonly used to implement communication in the client-server model: the client issues a request and the server responds to it. Figure 1-13 summarizes the types of services discussed above.

	Service	Example
Connection-oriented	Reliable message stream	Sequence of pages
	Reliable byte stream	Remote login
	Unreliable connection	Digitized voice
Connection-less	Unreliable datagram	Electronic junk mail
	Acknowledged datagram	Registered mail
	Request-reply	Database query

Fig. 1-13. Six different types of service.

1.3.5. Service Primitives

A service is formally specified by a set of **primitives** (operations) available to a user or other entity to access the service. These primitives tell the service to perform some action or report on an action taken by a peer entity. One way to classify the service primitives is to divide them into four classes as shown in Fig. 1-14.

Primitive	Meaning
Request	An entity wants the service to do some work
Indication	An entity is to be informed about an event
Response	An entity wants to respond to an event
Confirm	The response to an earlier request has come back

Fig. 1-14. Four classes of service primitives.

To illustrate the uses of the primitives, consider how a connection is established and released. The initiating entity does a CONNECT.request which results in a packet being sent. The receiver then gets a CONNECT.indication announcing that an entity somewhere wants to set up a connection to it. The entity getting the CONNECT.indication then uses the CONNECT.response primitive to tell whether it wants to accept or reject the proposed connection. Either way, the entity issuing the initial CONNECT.request finds out what happened via a CONNECT.confirm primitive.

Primitives can have parameters, and most of them do. The parameters to a CONNECT.request might specify the machine to connect to, the type of service desired, and the maximum message size to be used on the connection. The parameters to a CONNECT.indication might contain the caller's identity, the type of service desired, and the proposed maximum message size. If the called entity did not agree to the proposed maximum message size, it could make a counterproposal in its *response* primitive, which would be made available to the original caller in the *confirm*. The details of this **negotiation** are part of the protocol. For example, in the case of two conflicting proposals about maximum message size, the protocol might specify that the smaller value is always chosen.

As an aside on terminology, we will carefully avoid the terms "open a connection" and "close a connection" because to electrical engineers, an "open circuit" is one with a gap or break in it. Electricity can only flow over "closed circuits." Computer scientists would never agree to having information flow over a closed circuit. To keep both camps pacified, we will use the terms "establish a connection" and "release a connection."

Services can be either **confirmed** or **unconfirmed**. In a confirmed service, there is a *request*, an *indication*, a *response*, and a *confirm*. In an unconfirmed service, there is just a *request* and an *indication*. CONNECT is always a confirmed service because the remote peer must agree to establish a connection. Data transfer, on the other hand, can be either confirmed or unconfirmed, depending on whether or not the sender needs an acknowledgement. Both kinds of services are used in networks.

To make the concept of a service more concrete, let us consider as an example a simple connection-oriented service with eight service primitives as follows:

1. CONNECT.request – Request a connection to be established.
2. CONNECT.indication – Signal the called party.
3. CONNECT.response – Used by the callee to accept/reject calls.
4. CONNECT.confirm – Tell the caller whether the call was accepted.
5. DATA.request – Request that data be sent.
6. DATA.indication – Signal the arrival of data.
7. DISCONNECT.request – Request that a connection be released.
8. DISCONNECT.indication – Signal the peer about the request.

In this example, CONNECT is a confirmed service (an explicit response is required), whereas DISCONNECT is unconfirmed (no response).

It may be helpful to make an analogy with the telephone system to see how these primitives are used. For this analogy, consider the steps required to call Aunt Millie on the telephone and invite her to your house for tea.

1. CONNECT.request – Dial Aunt Millie's phone number.
2. CONNECT.indication – Her phone rings.
3. CONNECT.response – She picks up the phone.

4. CONNECT.confirm – You hear the ringing stop.
5. DATA.request – You invite her to tea.
6. DATA.indication – She hears your invitation.
7. DATA.request – She says she would be delighted to come.
8. DATA.indication – You hear her acceptance.
9. DISCONNECT.request – You hang up the phone.
10. DISCONNECT.indication – She hears it and hangs up too.

Figure 1-15 shows this same sequence of steps as a series of service primitives, including the final confirmation of disconnection. Each step involves an interaction between two layers on one of the computers. Each *request* or *response* causes an *indication* or *confirm* at the other side a little later. In this example, the service users (you and Aunt Millie) are in layer $N + 1$ and the service provider (the telephone system) is in layer N.

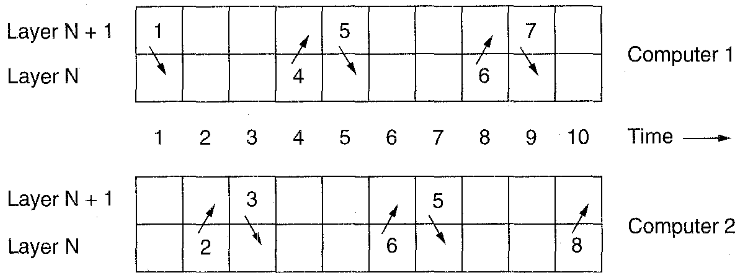


Fig. 1-15. How a computer would invite its Aunt Millie to tea. The numbers near the tail end of each arrow refer to the eight service primitives discussed in this section.

1.3.6. The Relationship of Services to Protocols

Services and protocols are distinct concepts, although they are frequently confused. This distinction is so important, however, that we emphasize it again here. A *service* is a set of primitives (operations) that a layer provides to the layer above it. The service defines what operations the layer is prepared to perform on behalf of its users, but it says nothing at all about how these operations are implemented. A service relates to an interface between two layers, with the lower layer being the service provider and the upper layer being the service user.

A *protocol*, in contrast, is a set of rules governing the format and meaning of the frames, packets, or messages that are exchanged by the peer entities within a layer. Entities use protocols in order to implement their service definitions. They 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.

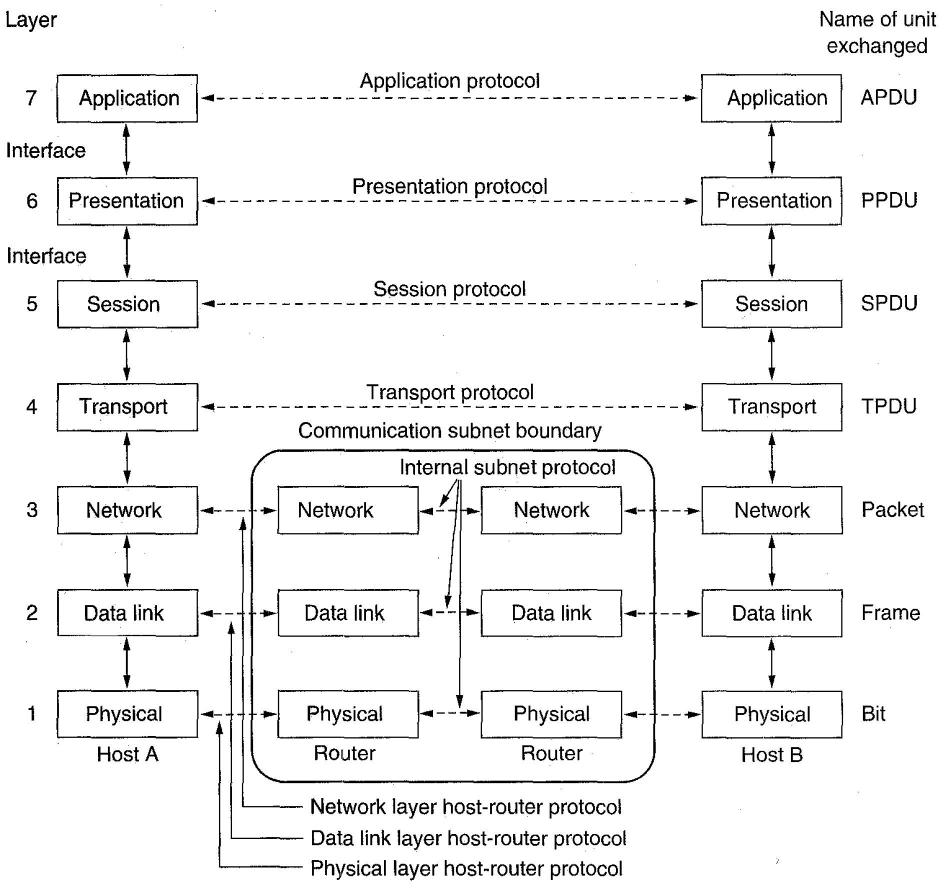


Fig. 1-16. The OSI reference model.

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.

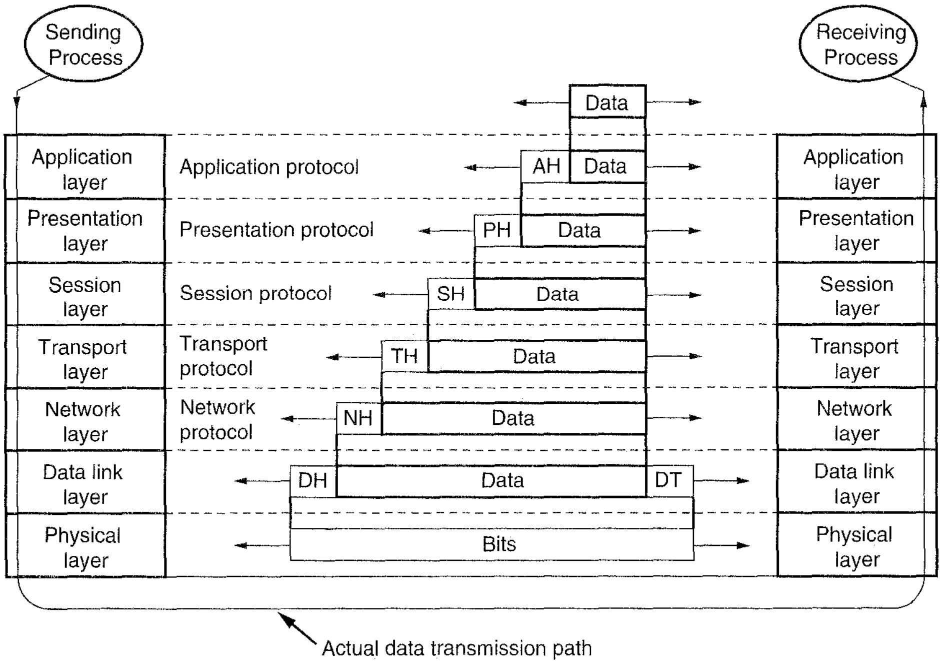


Fig. 1-17. An example of how the OSI model is used. Some of the headers may be null. (Source: H.C. Folts. Used with permission.)

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.

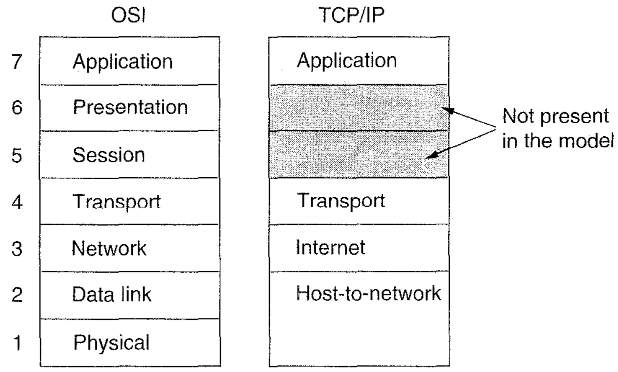


Fig. 1-18. The TCP/IP reference model.

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.

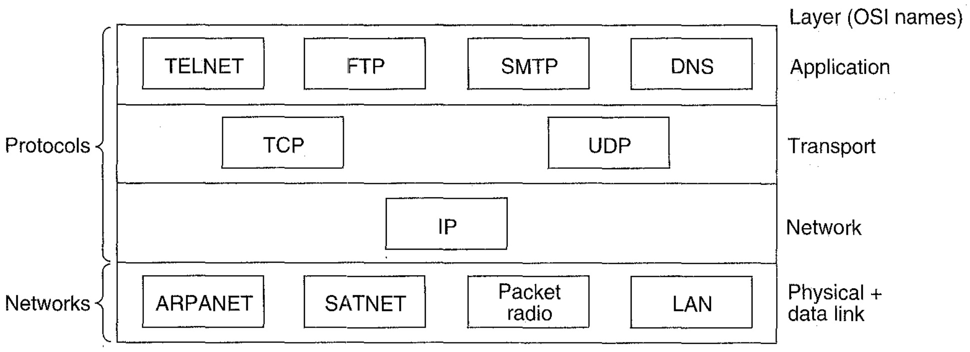


Fig. 1-19. Protocols and networks in the TCP/IP model initially.

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 Wide, and many others.

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.

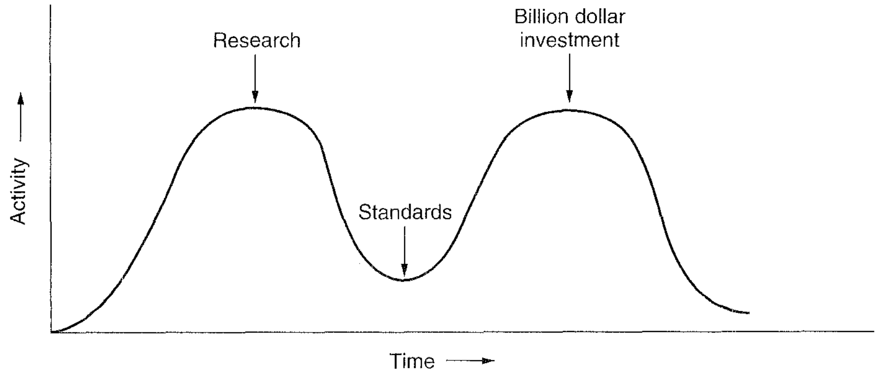


Fig. 1-20. The apocalypse of the two elephants.

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 at a few current and historical networks to get an idea of what they are (or were) like and how they differ from one another.

Networks differ in their history, administration, facilities offered, technical design, and user communities. The history and administration can vary from a network carefully planned by a single organization with a well-defined goal, to an ad hoc collection of machines that have been connected to one another over the years without any master plan or central administration at all. The facilities available range from arbitrary process-to-process communication to electronic mail, file transfer, remote login, and remote execution. The technical designs can differ in the transmission media used, the naming and routing algorithms employed, the number and contents of the layers present, and the protocols used. Finally, the user community can vary from a single corporation to all the academic computer scientists in the industrialized world.

In the following sections we will look at a few examples. These are the popular commercial LAN networking package, Novell NetWare®, the worldwide Internet (including its predecessors, the ARPANET and NSFNET), and the first gigabit networks.

1.5.1. Novell NetWare

The most popular network system in the PC world is **Novell NetWare**. It was designed to be used by companies downsizing from a mainframe to a network of PCs. In such systems, each user has a desktop PC functioning as a client. In addition, some number of powerful PCs operate as servers, providing file services, database services, and other services to a collection of clients. In other words, Novell NetWare is based on the client-server model.

NetWare uses a proprietary protocol stack illustrated in Fig. 1-22. It is based on the old Xerox Network System, XNS™ but with various modifications. Novell NetWare predates OSI and is not based on it. If anything, it looks more like TCP/IP than like OSI.

Layer			
Application	SAP	File server	. . .
Transport	NCP		SPX
Network	IPX		
Data link	Ethernet	Token ring	ARCnet
Physical	Ethernet	Token ring	ARCnet

Fig. 1-22. The Novell NetWare reference model.

The physical and data link layers can be chosen from among various industry standards, including Ethernet, IBM token ring, and ARCnet. The network layer runs an unreliable connectionless internetwork protocol called **IPX** (**Internet Packet eXchange**). It passes packets transparently from source to destination, even if the source and destination are on different networks. IPX is functionally similar to IP, except that it uses 12-byte addresses instead of 4-byte addresses. The wisdom of this choice will become apparent in Chap. 5.

Above IPX comes a connection-oriented transport protocol called **NCP** (**Network Core Protocol**). NCP also provides various other services besides user data transport and is really the heart of NetWare. A second protocol, **SPX** (**Sequenced Packet eXchange**), is also available, but provides only transport. TCP is another option. Applications can choose any of them. The file system uses NCP and Lotus Notes® uses SPX, for example. The session and presentation layers do not exist. Various application protocols are present in the application layer.

As in TCP/IP, the key to the entire architecture is the internet datagram packet on top of which everything else is built. The format of an IPX packet is shown in Fig. 1-23. The *Checksum* field is rarely used, since the underlying data link layer also provides a checksum. The *Packet length* field tells how long the entire packet is, header plus data. The *Transport control* field counts how many networks the packet has traversed. When this exceeds a maximum, the packet is discarded. The *Packet type* field is used to mark various control packets. The two addresses each contain a 32-bit network number, a 48-bit machine number (the 802 LAN address), and 16-bit local address (socket) on that machine. Finally, we have the data, which occupy the rest of the packet, with the maximum size being determined by the underlying network.

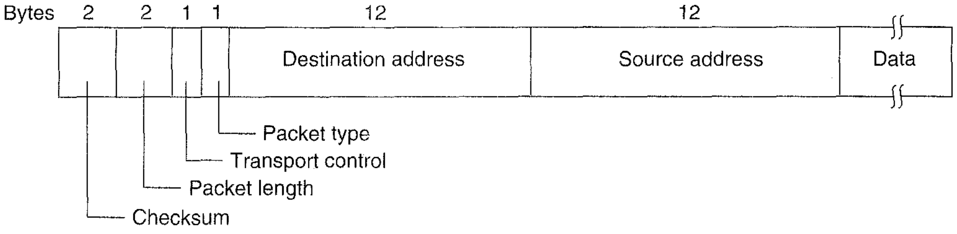


Fig. 1-23. A Novell NetWare IPX packet.

About once a minute, each server broadcasts a packet giving its address and telling what services it offers. These broadcasts use the **SAP** (**Service Advertising Protocol**) protocol. The packets are seen and collected by special agent processes running on the router machines. The agents use the information contained in them to construct databases of which servers are running where.

When a client machine is booted, it broadcasts a request asking where the nearest server is. The agent on the local router machine sees this request, looks in its database of servers, and matches up the request with the best server. The choice of server to use is then sent back to the client. The client can now establish an NCP connection with the server. Using this connection, the client and server negotiate the maximum packet size. From this point on, the client can access the file system and other services using this connection. It can also query the server's database to look for other (more distant) servers.

1.5.2. The ARPANET

Let us now switch gears from LANs to WANs. In the mid-1960s, at the height of the Cold War, the DoD wanted a command and control network that could survive a nuclear war. Traditional circuit-switched telephone networks were considered too vulnerable, since the loss of one line or switch would certainly terminate all conversations using them and might even partition the network. To solve this problem, DoD turned to its research arm, ARPA (later DARPA, now ARPA again), the (periodically Defense) Advanced Research Projects Agency.

ARPA was created in response to the Soviet Union's launching Sputnik in 1957 and had the mission of advancing technology that might be useful to the military. ARPA had no scientists or laboratories, in fact, it had nothing more than an office and a small (by Pentagon standards) budget. It did its work by issuing grants and contracts to universities and companies whose ideas looked promising to it.

Several early grants went to universities for investigating the then-radical idea of packet switching, something that had been suggested by Paul Baran in a series of RAND Corporation reports published in the early 1960s. After some discussions with various experts, ARPA decided that the network the DoD needed should be a packet-switched network, consisting of a subnet and host computers.

The subnet would consist of minicomputers called **IMPs** (**Interface Message Processors**) connected by transmission lines. For high reliability, each IMP would be connected to at least two other IMPs. The subnet was to be a datagram subnet, so if some lines and IMPs were destroyed, messages could be automatically rerouted along alternative paths.

Each node of the network was to consist of an IMP and a host, in the same room, connected by a short wire. A host could send messages of up to 8063 bits to its IMP, which would then break these up into packets of at most 1008 bits and forward them independently toward the destination. Each packet was received in its entirety before being forwarded, so the subnet was the first electronic store-and-forward packet-switching network.

ARPA then put out a tender for building the subnet. Twelve companies bid for it. After evaluating all the proposals, ARPA selected BBN, a consulting firm in Cambridge, Massachusetts, and in December 1968, awarded it a contract to build the subnet and write the subnet software. BBN chose to use specially modified Honeywell DDP-316 minicomputers with 12K 16-bit words of core memory as the IMPs. The IMPs did not have disks, since moving parts were considered unreliable. The IMPs were interconnected by 56-kbps lines leased from telephone companies.

The software was split into two parts: subnet and host. The subnet software consisted of the IMP end of the host-IMP connection, the IMP-IMP protocol, and a source IMP to destination IMP protocol designed to improve reliability. The original ARPANET design is shown in Fig. 1-24.

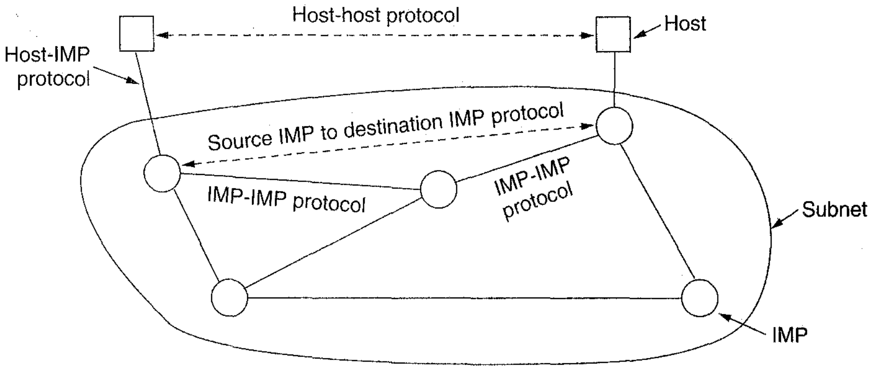


Fig. 1-24. The original ARPANET design.

Outside the subnet, software was also needed, namely, the host end of the host-IMP connection, the host-host protocol, and the application software. It soon became clear that BBN felt that when it had accepted a message on a host-IMP wire and placed it on the host-IMP wire at the destination, its job was done.

To deal with problem of host software, Larry Roberts of ARPA convened a meeting of network researchers, mostly graduate students, at Snowbird, Utah, in the summer of 1969. The graduate students expected some network expert to explain the design of the network and its software to them and then to assign each of them the job of writing part of it. They were astounded when there was no network expert and no grand design. They had to figure out what to do on their own.

Nevertheless, somehow an experimental network went on the air in December 1969 with four nodes, at UCLA, UCSB, SRI, and the University of Utah. These four were chosen because all had a large number of ARPA contracts, and all had different and completely incompatible host computers (just to make it more fun). The network grew quickly as more IMPs were delivered and installed; it soon spanned the United States. Figure 1-25 shows how rapidly the ARPANET grew in the first 3 years.

Later the IMP software was changed to allow terminals to connect directly to a special IMP, called a **TIP** (**Terminal Interface Processor**), without having to go through a host. Subsequent changes included having multiple hosts per IMP (to save money), hosts talking to multiple IMPs (to protect against IMP failures),

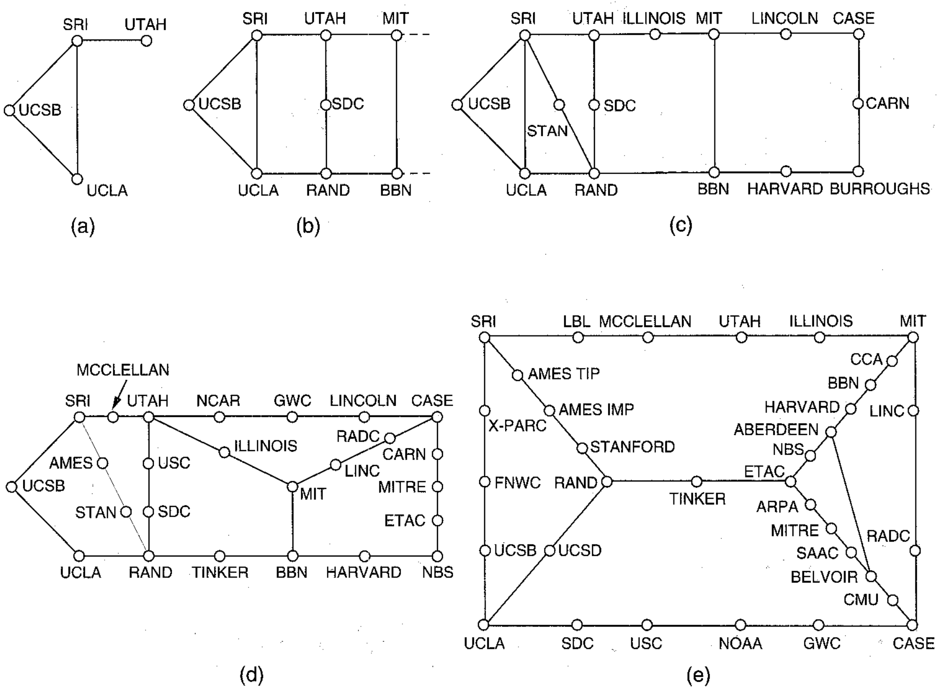


Fig. 1-25. Growth of the ARPANET. (a) Dec. 1969. (b) July 1970. (c) March 1971. (d) April 1972. (e) Sept. 1972.

and hosts and IMPs separated by a large distance (to accommodate hosts far from the subnet).

In addition to helping the fledgling ARPANET grow, ARPA also funded research on satellite networks and mobile packet radio networks. In one famous demonstration, a truck driving around in California used the packet radio network to send messages to SRI, which were then forwarded over the ARPANET to the East Coast, where they were shipped to University College in London over the satellite network. This allowed a researcher in the truck to use a computer in London while driving around in California.

This experiment also demonstrated that the existing ARPANET protocols were not suitable for running over multiple networks. This observation led to more research on protocols, culminating with the invention of the TCP/IP model and protocols (Cerf and Kahn, 1974). TCP/IP was specifically designed to handle communication over internetworks, something becoming increasingly important as more and more networks were being hooked up to the ARPANET.

To encourage adoption of these new protocols, ARPA awarded several contracts to BBN and the University of California at Berkeley to integrate them into Berkeley UNIX. Researchers at Berkeley developed a convenient program interface to the network (sockets) and wrote many application, utility, and management programs to make networking easier.

The timing was perfect. Many universities had just acquired a second or third VAX computer and a LAN to connect them, but they had no networking software. When 4.2BSD came along, with TCP/IP, sockets, and many network utilities, the complete package was adopted immediately. Furthermore, with TCP/IP, it was easy for the LANs to connect to the ARPANET, and many did.

By 1983, the ARPANET was stable and successful, with over 200 IMPs and hundreds of hosts. At this point, ARPA turned the management of the network over to the Defense Communications Agency (DCA), to run it as an operational network. The first thing DCA did was to separate the military portion (about 160 IMPs, of which 110 in the United States and 50 abroad) into a separate subnet, **MILNET**, with stringent gateways between MILNET and the remaining research subnet.

During the 1980s, additional networks, especially LANs, were connected to the ARPANET. As the scale increased, finding hosts became increasingly expensive, so **DNS** (**Domain Naming System**) was created to organize machines into domains and map host names onto IP addresses. Since then, DNS has become a generalized, distributed database system for storing a variety of information related to naming. We will study it in detail in Chap. 7.

By 1990, the ARPANET had been overtaken by newer networks that it itself had spawned, so it was shut down and dismantled, but it lives on in the hearts and minds of network researchers everywhere. MILNET continues to operate, however.

1.5.3. NSFNET

By the late 1970s, NSF (the U.S. National Science Foundation) saw the enormous impact the ARPANET was having on university research, allowing scientists across the country to share data and collaborate on research projects. However, to get on the ARPANET, a university had to have a research contract with the DoD, which many did not have. This lack of universal access prompted NSF to set up a virtual network, **CSNET**, centered around a single machine at BBN that supported dial-up lines and had connections to the ARPANET and other networks. Using CSNET, academic researchers could call up and leave email for other people to pick up later. It was simple, but it worked.

By 1984 NSF began designing a high-speed successor to the ARPANET that would be open to all university research groups. To have something concrete to start with, NSF decided to build a backbone network to connect its six supercomputer centers, in San Diego, Boulder, Champaign, Pittsburgh, Ithaca, and Princeton. Each supercomputer was given a little brother, consisting of an LSI-11 microcomputer called a **fuzzball**. The fuzzballs were connected with 56 kbps leased lines and formed the subnet, the same hardware technology as the

ARPANET used. The software technology was different however: the fuzzballs spoke TCP/IP right from the start, making it the first TCP/IP WAN.

NSF also funded some (eventually about 20) regional networks that connected to the backbone to allow users at thousands of universities, research labs, libraries, and museums to access any of the supercomputers and to communicate with one another. The complete network, including the backbone and the regional networks, was called **NSFNET**. It connected to the ARPANET through a link between an IMP and a fuzzball in the Carnegie-Mellon machine room. The first NSFNET backbone is illustrated in Fig. 1-26.

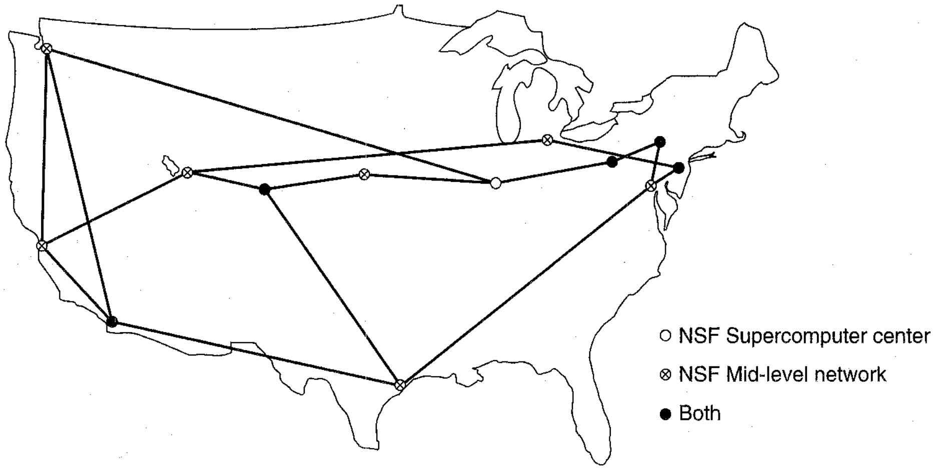


Fig. 1-26. The NSFNET backbone in 1988.

NSFNET was an instantaneous success and was overloaded from the word go. NSF immediately began planning its successor and awarded a contract to the Michigan-based MERIT consortium to run it. Fiber optic channels at 448 kbps were leased from MCI to provide the version 2 backbone. IBM RS6000s were used as routers. This, too, was soon overwhelmed, and by 1990, the second backbone was upgraded to 1.5 Mbps.

As growth continued, NSF realized that the government could not continue financing networking forever. Furthermore, commercial organizations wanted to join but were forbidden by NSF's charter from using networks NSF paid for. Consequently, NSF encouraged MERIT, MCI, and IBM to form a nonprofit corporation, **ANS** (**Advanced Networks and Services**) as a step along the road to commercialization. In 1990, ANS took over NSFNET and upgraded the 1.5-Mbps links to 45 Mbps to form **ANSNET**.

In December 1991, the U.S. Congress passed a bill authorizing **NREN**, the **National Research and Educational Network**, the research successor to NSFNET, only running at gigabits speeds. The goal was a national network running at 3 Gbps before the millenium. This network is to act as a prototype for the much-discussed information superhighway.

By 1995, the NSFNET backbone was no longer needed to interconnect the NSF regional networks because numerous companies were running commercial IP networks. When ANSNET was sold to America Online in 1995, the NSF regional networks had to go out and buy commercial IP service to interconnect.

To ease the transition and make sure every regional network could communicate with every other regional network, NSF awarded contracts to four different network operators to establish a **NAP** (**Network Access Point**). These operators were PacBell (San Francisco), Ameritech (Chicago), MFS (Washington, D.C.), and Sprint (New York City, where for NAP purposes, Pennsauken, N.J. counts as New York City). Every network operator that wanted to provide backbone service to the NSF regional networks had to connect to all the NAPs. This arrangement meant that a packet originating on any regional network had a choice of backbone carriers to get from its NAP to the destination's NAP. Consequently, the backbone carriers were forced to compete for the regional networks' business on the basis of service and price, which was the idea, of course. In addition to the NSF NAPs, various government NAPs (e.g., FIX-E, FIX-W, MAE-East and MAE-West) and commercial NAPs (e.g., CIX) have also been created, so the concept of a single default backbone was replaced by a commercially-driven competitive infrastructure.

Other countries and regions are also building networks comparable to NSFNET. In Europe, for example, EBONE is an IP backbone for research organizations and EuropaNET is a more commercially oriented network. Both connect numerous cities in Europe with 2-Mbps lines. Upgrades to 34 Mbps are in progress. Each country in Europe has one or more national networks, which are roughly comparable to the NSF regional networks.

1.5.4. The Internet

The number of networks, machines, and users connected to the ARPANET grew rapidly after TCP/IP became the only official protocol on Jan. 1, 1983. When NSFNET and the ARPANET were interconnected, the growth became exponential. Many regional networks joined up, and connections were made to networks in Canada, Europe, and the Pacific.

Sometime in the mid-1980s, people began viewing the collection of networks as an internet, and later as the Internet, although there was no official dedication with some politician breaking a bottle of champagne over a fuzzball.

Growth continued exponentially, and by 1990 the Internet had grown to 3000 networks and 200,000 computers. In 1992, the one millionth host was attached. By 1995, there were multiple backbones, hundreds of mid-level (i.e., regional) networks, tens of thousands of LANs, millions of hosts, and tens of millions of users. The size doubles approximately every year (Paxson, 1994).

Much of the growth comes from connecting existing networks to the Internet. In the past these have included SPAN, NASA's space physics network, HEPNET, a high energy physics network, BITNET, IBM's mainframe network, EARN, a European academic network now widely used in Eastern Europe, and many others. Numerous transatlantic links are in use, running from 64 kbps to 2 Mbps.

The glue that holds the Internet together is the TCP/IP reference model and TCP/IP protocol stack. TCP/IP makes universal service possible and can be compared to the telephone system or the adoption of standard gauge by the railroads in the 19th Century.

What does it actually mean to be on the Internet? Our definition is that a machine is on the Internet if it runs the TCP/IP protocol stack, has an IP address, and has the ability to send IP packets to all the other machines on the Internet. The mere ability to send and receive electronic mail is not enough, since email is gatewayed to many networks outside the Internet. However, the issue is clouded somewhat by the fact that many personal computers have the ability to call up an Internet service provider using a modem, be assigned a temporary IP address, and send IP packets to other Internet hosts. It make sense to regard such machines as being on the Internet for as long as they are connected to the service provider's router.

With exponential growth, the old informal way of running the Internet no longer works. In January 1992, the **Internet Society** was set up, to promote the use of the Internet and perhaps eventually take over managing it.

Traditionally, the Internet had four main applications, as follows:

1. **Email**. The ability to compose, send, and receive electronic mail has been around since the early days of the ARPANET and is enormously popular. Many people get dozens of messages a day and consider it their primary way of interacting with the outside world, far outdistancing the telephone and snail mail. Email programs are available on virtually every kind of computer these days.
2. **News**. Newsgroups are specialized forums in which users with a common interest can exchange messages. Thousands of newsgroups exist, on technical and nontechnical topics, including computers, science, recreation, and politics. Each newsgroup has its own etiquette, style, and customs, and woe be to anyone violating them.
3. **Remote login**. Using the Telnet, Rlogin, or other programs, users anywhere on the Internet can log into any other machine on which they have an account.
4. **File transfer**. Using the FTP program, it is possible to copy files from one machine on the Internet to another. Vast numbers of articles, databases, and other information are available this way.

Up until the early 1990s, the Internet was largely populated by academic, government, and industrial researchers. One new application, the **WWW** (**World Wide Web**) changed all that and brought millions of new, nonacademic users to the net. This application, invented by CERN physicist Tim Berners-Lee, did not change any of the underlying facilities but made them easier to use. Together with the Mosaic viewer, written at the National Center for Supercomputer Applications, the WWW made it possible for a site to set up a number of pages of information containing text, pictures, sound, and even video, with embedded links to other pages. By clicking on a link, the user is suddenly transported to the page pointed to by that link. For example, many companies have a home page with entries pointing to other pages for product information, price lists, sales, technical support, communication with employees, stockholder information, and much more.

Numerous other kinds of pages have come into existence in a very short time, including maps, stock market tables, library card catalogs, recorded radio programs, and even a page pointing to the complete text of many books whose copyrights have expired (Mark Twain, Charles Dickens, etc.). Many people also have personal pages (home pages).

In the first year after Mosaic was released, the number of WWW servers grew from 100 to 7000. Enormous growth will undoubtedly continue for years to come, and will probably be the force driving the technology and use of the Internet into the next millenium.

Many books have been written about the Internet and its protocols. For more information, see (Black, 1995; Carl-Mitchell and Quarterman, 1993; Comer, 1995; and Santifaller, 1994).

1.5.5. Gigabit Testbeds

The Internet backbones operate at megabit speeds, so for people who want to push the technological envelope, the next step is gigabit networking. With each increase in network bandwidth, new applications become possible, and gigabit networks are no exception. In this section we will first say a few words about gigabit applications, mention two of them, and then list some example gigabit testbeds that have been built.

Gigabit networks provide better bandwidth than megabit networks, but not always much better delay. For example, sending a 1-kbit packet from New York to San Francisco at 1 Mbps takes 1 msec to pump the bits out and 20 msec for the transcontinental delay, for a total of 21 msec. A 1-Gbps network can reduce this to 20.001 msec. While the bits go out faster, the transcontinental delay remains the same, since the speed of light in optical fiber (or copper wire) is about 200,000 km/sec, independent of the data rate. Thus for wide area applications in which low delay is critical, going to higher speeds may not help much. Fortunately, for some applications, bandwidth is what counts, and these are the applications for which gigabit networks will make a big difference.

One application is telemedicine. Many people think that a way to reduce medical costs is to reintroduce family doctors and family clinics on a large scale, so everyone has convenient access to first line medical care. When a serious medical problem occurs, the family doctor can order lab tests and medical imaging, such as X-rays, CAT scans, and MRI scans. The test results and images can then be sent electronically to a specialist who then makes the diagnosis.

Doctors are generally unwilling to make diagnoses from computer images unless the quality of the transmitted image is as good as the original image. This requirement means images will probably need 4K × 4K pixels, with 8 bits per pixel (black and white images) or 24 bits per pixel (color images). Since many tests require up to 100 images (e.g., different cross sections of the organ in question), a single series for one patient can generate 40 gigabits. Moving images (e.g., a beating heart) generate even more data. Compression can help some but doctors are leary of it because the most efficient algorithms reduce image quality. Furthermore, all the images must be stored for years but may need to be retrieved at a moment's notice in the event of a medical emergency. Hospitals do not want to become computer centers, so off-site storage combined with high-bandwidth electronic retrieval is essential.

Another gigabit application is the virtual meeting. Each meeting room contains a spherical camera and one or more people. The bit streams from each of the cameras are combined electronically to give the illusion that everyone is in the same room. Each person sees this image using virtual reality goggles. In this way meetings can happen without travel, but again, the data rates required are stupendous.

Starting in 1989, ARPA and NSF jointly agreed to finance a number of university-industry gigabit testbeds, later as part of the NREN project. In some of these, the data rate in each direction was 622 Mbps, so only by counting the data going in both directions do you get a gigabit. This kind of gigabit is sometimes called a "government gigabit." (Some cynics call it a gigabit after taxes.) Below we will briefly mention the first five projects. They have done their job and been shut down, but deserve some credit as pioneers, in the same way the ARPANET does.

1. **Aurora** was a testbed linking four sites in the Northeast: M.I.T., the University of Pennsylvania, IBM's T.J. Watson Lab, and Bellcore (Morristown, N.J.) at 622 Mbps using fiber optics provided by MCI, Bell Atlantic, and NYNEX. Aurora was largely designed to help debug Bellcore's Sunshine switch and IBM's (proprietary) plaNET switch using parallel networks. Research issues included switching technology, gigabit protocols, routing, network control, distributed virtual memory, and collaboration using videoconferencing. For more information, see (Clark et al., 1993).

2. **Blanca** was originally a research project called XUNET involving AT&T Bell Labs, Berkeley, and the University of Wisconsin. In 1990 it added some new sites (LBL, Cray Research, and the University of Illinois) and acquired NSF/ARPA funding. Some of it ran at 622 Mbps, but other parts ran at lower speeds. Blanca was the only nationwide testbed; the rest were regional. Consequently, much of the research was concerned with the effects of speed-of-light delay. The interest here was in protocols, especially network control protocols, host interfaces, and gigabit applications such as medical imaging, meteorological modeling, and radio astronomy. For more information, see (Catlett, 1992; and Fraser, 1993).
3. **CASA** was aimed at doing research on supercomputer applications, especially those in which part of the problem ran best on one kind of supercomputer (e.g., a Cray vector supercomputer) and part ran best on a different kind of supercomputer (e.g., a parallel supercomputer). The applications investigated included geology (analyzing Landsat images), climate modeling, and understanding chemical reactions. It operated in California and New Mexico and connected Los Alamos, Cal Tech, JPL, and the San Diego Supercomputer Center.
4. **Nectar** differed from the three testbeds given above in that it was an experimental gigabit MAN running from CMU to the Pittsburgh Supercomputer Center. The designers were interested in applications involving chemical process flowsheeting and operations research, as well as the tools for debugging them.
5. **VISTAnet** was a small gigabit testbed operated in Research Triangle Park, North Carolina, and connecting the University of North Carolina, North Carolina State University, and MCNC. The interest here was in a prototype for a public switched gigabit network with switches having hundreds of gigabit lines, meaning that the switches had to be capable of processing terabits/sec. The scientific research focused on using 3D images to plan radiation therapy for cancer patients, with the oncologist being able to vary the beam parameters and instantaneously see the radiation dosages being delivered to the tumor and surrounding tissue (Ransom, 1992).

1.6. EXAMPLE DATA COMMUNICATION SERVICES

Telephone companies and others have begun to offer networking services to any organization that wishes to subscribe. The subnet is owned by the network operator, providing communication service for the customers' hosts and terminals.

Such a system is called a **public network**. It is analogous to, and often a part of, the public telephone system. We already briefly looked at one new service, DQDB, in Fig. 1-4. In the following sections we will study four other example services, SMDS, X.25, frame relay, and broadband ISDN.

1.6.1. SMDS—Switched Multimegabit Data Service

The first service we will look at, **SMDS** (**Switched Multimegabit Data Service**), was designed to connect together multiple LANs, typically at the branch offices and factories of a single company. It was designed by Bellcore in the 1980s and deployed in the early 1990s by regional and a few long distance carriers. The goal was to produce a high-speed data service and get it out into the world with a minimum of fuss. SMDS is the first broadband (i.e., high-speed) switched service offered to the public.

To see a situation in which SMDS would be useful, consider a company with four offices in four different cities, each with its own LAN. The company would like to connect all the LANs, so that packets can go from one LAN to another. One solution would be to lease six high-speed lines and fully connect the LANs as shown in Fig. 1-27(a). Such a solution is certainly possible, but expensive.

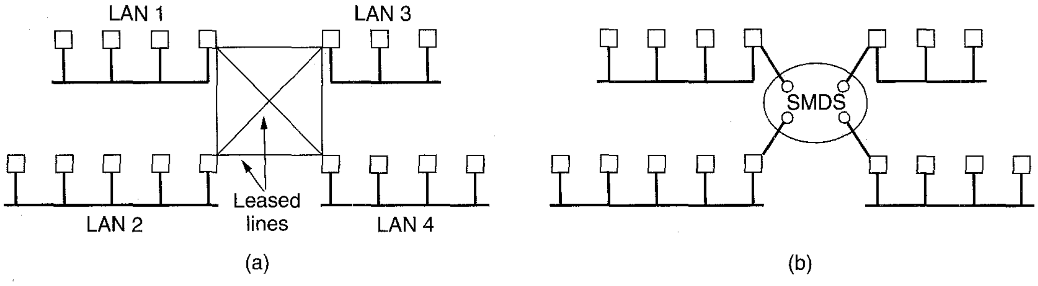


Fig. 1-27. (a) Four LANs interconnected with leased lines. (b) Interconnection using SMDS.

An alternative solution is to use SMDS, as shown in Fig. 1-27(b). The SMDS network acts like a high-speed LAN backbone, allowing packets from any LAN to flow to any other LAN. Between the LANs, in the customer's offices, and the SMDS network, in the telephone company's offices, is a (short) access line leased from the telephone company. Usually, this line is a MAN and uses DQDB, but other options may also be available.

Whereas most telephone company services are designed for continuous traffic, SMDS is designed to handle bursty traffic. In other words, once in a while a packet has to be carried from one LAN to another quickly, but much of the time there is no LAN to LAN traffic. The leased line solution of Fig. 1-27(a) has the problem of high monthly bills; once installed, the customer has to pay for the lines whether or not they are used continuously. For intermittent traffic, leased lines are an expensive solution, and SMDS is priced to compete with them. With n LANs, a fully connected leased line network requires leasing $n(n - 1)/2$ possibly long (i.e., expensive) lines, whereas SMDS only requires leasing n short access lines to the nearest SMDS router.

Since the goal of SMDS is to carry LAN to LAN traffic, it must be fast enough to do the job. The standard speed is 45 Mbps, although sometimes lower speed options are available. MANs can also operate at 45 Mbps, but they are not switched, that is, to connect four LANs using a MAN, the telephone company would have to run a single wire from LAN 1 to LAN 2 to LAN 3 to LAN 4, which is only possible if they are in the same city. With SMDS, each LAN connects to a telephone company switch which routes packets through the SMDS network as needed to reach the destination, possibly traversing multiple switches in the process.

The basic SMDS service is a simple connectionless packet delivery service. The packet format is shown in Fig. 1-28. It has three fields: the destination (where the packet is to go to), the source (who sent it), and a variable length payload field for up to 9188 bytes of user data. The machine on the sending LAN that is connected to the access line puts the packet on the access line, and SMDS makes a best effort attempt to deliver it to the correct destination. No guarantee is given.

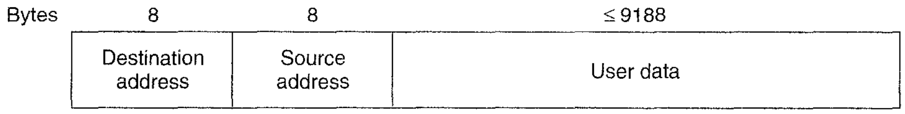


Fig. 1-28. The SMDS packet format.

The source and destination addresses consist of a 4-bit code followed by a telephone number of up to 15 decimal digits. Each digit is coded in a separate 4-bit field. The telephone numbers contain country code, area code, and subscriber number, so the service could eventually be offered internationally. It was thought that having decimal telephone numbers as network addresses would make the new offering seem familiar to nervous users.

When a packet arrives at the SMDS network, the first router checks to make sure that the source address corresponds to the incoming line, to prevent billing fraud. If the address is incorrect, the packet is simply discarded. If it is correct, the packet is sent along toward its destination.

A useful SMDS feature is broadcasting. The customer can specify a list of SMDS telephone numbers, and be assigned a special number for the whole list. Any packet sent to that number is delivered to all members on that list. The National Association of Securities Dealers uses this feature of MCI's SMDS service to broadcast new stock prices to all of its 5000 members.

An additional user feature is address screening, on both outgoing and incoming packets. With outgoing screening, the customer can give a list of telephone numbers and specify that no packets may be sent to any other addresses. With incoming screening, only packets from certain pre-arranged telephone numbers will be accepted. When both features are enabled, the user can effectively build a private network with no SMDS connections to the outside world. For companies with confidential data, this feature is highly valuable.

The payload can contain any byte sequence the user wishes, up to 9188 bytes. SMDS does not look at it. It can contain an Ethernet packet, an IBM token ring packet, an IP packet, or anything else. Whatever is present in the payload field is moved without modification from the source LAN to the destination LAN.

SMDS handles bursty traffic as follows. The router connected to each access line contains a counter that is incremented at a constant rate, say once every 10 μsec. When a packet arrives at the router, a check is made to see if the counter is greater than the packet length, in bytes. If it is, the packet is sent without delay and the counter is decremented by the packet length. If the packet length is greater than the counter, the packet is discarded.

In effect, with a tick every 10 μsec the user may send at an *average* rate of 100,000 bytes/sec, but the burst rate may be much higher. If, for example, the line has been idle for 10 msec, the counter will be 1000, and the user will be allowed to send a 1-kilobyte burst at the full 45 Mbps, so it will be transmitted in about 180 μsec. With a 100,000 byte/sec leased line, the same kilobyte would take 10 msec. Thus SMDS offers short delays for widely spaced independent data bursts, as long as the average rate remains below the agreed upon value. This mechanism provides fast response when needed but prevents users from using up more bandwidth than they have agreed to pay for.

1.6.2. X.25 Networks

Many older public networks, especially outside the United States, follow a standard called **X.25**. It was developed during the 1970s by CCITT to provide an interface between public packet-switched networks and their customers.

The physical layer protocol, called **X.21**, specifies the physical, electrical, and procedural interface between the host and the network. Very few public networks actually support this standard, because it requires digital, rather than analog signaling on the telephone lines. As an interim measure, an analog interface similar to the familiar RS-232 standard was defined.

The data link layer standard has a number of (slightly incompatible) variations. They all are designed to deal with transmission errors on the telephone line between the user's equipment (host or terminal) and the public network (router).

The network layer protocol deals with addressing, flow control, delivery confirmation, interrupts, and related issues. Basically, it allows the user to establish virtual circuits and then send packets of up to 128 bytes on them. These packets are delivered reliably and in order. Most X.25 networks work at speeds up to 64 kbps, which makes them obsolete for many purposes. Nevertheless, they are still widespread, so readers should be aware of their existence.

X.25 is connection-oriented and supports both switched virtual circuits and permanent ones. A **switched virtual circuit** is created when one computer sends a packet to the network asking to make a call to a remote computer. Once established, packets can be sent over the connection, always arriving in order. X.25 provides flow control, to make sure a fast sender cannot swamp a slow or busy receiver.

A **permanent virtual circuit** is used the same way as a switched one, but it is set up in advance by agreement between the customer and the carrier. It is always present, and no call setup is required to use it. It is analogous to a leased line.

Because the world is still full of terminals that do not speak X.25, another set of standards was defined that describes how an ordinary (nonintelligent) terminal communicates with an X.25 public network. In effect, the user or network operator installs a "black box" to which these terminals can connect. The black box is called a **PAD** (**Packet Assembler Disassembler**), and its function is described in a document known as **X.3**. A standard protocol has been defined between the terminal and the PAD, called **X.28**; another standard protocol exists between the PAD and the network, called **X.29**. Together, these three recommendations are often called **triple X**.

1.6.3. Frame Relay

Frame relay is a service for people who want an absolute bare-bones connection-oriented way to move bits from *A* to *B* at reasonable speed and low cost (Smith, 1993). Its existence is due to changes in technology over the past two decades. Twenty years ago, communication using telephone lines was slow, analog, and unreliable, and computers were slow and expensive. As a result, complex protocols were required to mask errors, and the users' computers were too expensive to have them do this work.

The situation has changed radically. Leased telephone lines are now fast, digital, and reliable, and computers are fast and inexpensive. This suggests the use of simple protocols, with most of the work being done by the users' computers, rather than by the network. It is this environment that frame relay addresses.

Frame relay can best be thought of as a virtual leased line. The customer leases a permanent virtual circuit between two points and can then send frames (i.e., packets) of up to 1600 bytes between them. It is also possible to lease permanent virtual circuits between a given site and multiple other sites, so each frame carries a 10-bit number telling which virtual circuit to use.

The difference between an actual leased line and a virtual leased line is that with an actual one, the user can send traffic all day long at the maximum speed. With a virtual one, data bursts may be sent at full speed, but the long-term average 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.

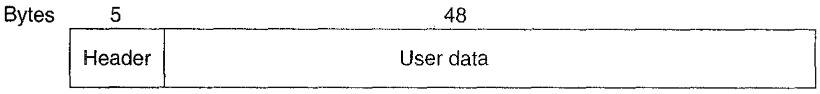


Fig. 1-29. An ATM cell.

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.

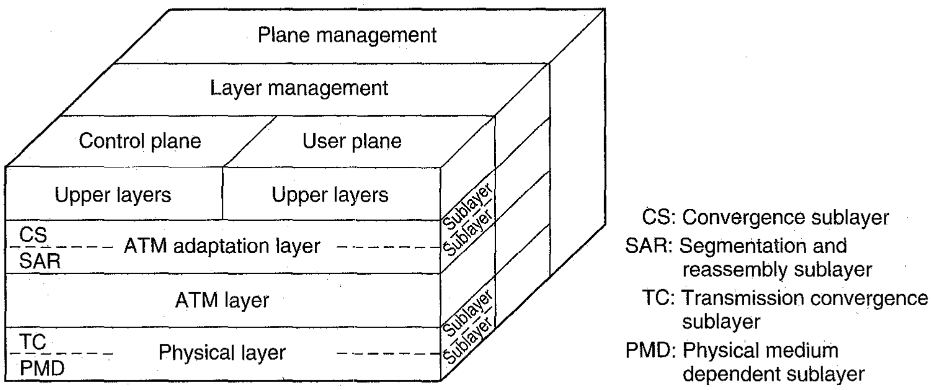


Fig. 1-30. The B-ISDN ATM reference model.

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 mass production, economies of scale in manufacturing, VLSI implementations, and other benefits that decrease price and further increase acceptance. In the following sections we will take a quick look at the important, but little-known, world of international standardization.

Standards fall into two categories: de facto and de jure. **De facto** (Latin for "from the fact") standards are those that have just happened, without any formal plan. The IBM PC and its successors are de facto standards for small office computers because dozens of manufacturers have chosen to copy IBM's machines very closely. UNIX is the de facto standard for operating systems in university computer science departments.

De jure (Latin for "by law") standards, in contrast, are formal, legal standards adopted by some authorized standardization body. International standardization authorities are generally divided into two classes: those established by treaty among national governments, and voluntary, nontreaty organizations. In the area of computer network standards, there are several organizations of each type, which are discussed below.

1.7.1. Who's Who in the Telecommunications World

The legal status of the world's telephone companies varies considerably from country to country. At one extreme is the United States, which has 1500 separate, privately owned telephone companies. Before it was broken up in 1984, AT&T, at that time the world's largest corporation, completely dominated the scene. It provided telephone service to about 80 percent of America's telephones, spread throughout half of its geographical area, with all the other companies combined servicing the remaining (mostly rural) customers. Since the breakup, AT&T continues to provide long-distance service, although now in competition with other companies. The seven Regional Bell Operating Companies that were split off from AT&T and 1500 independents provide local and cellular telephone service. Some of these independents, such as GTE, are very large companies.

Companies in the United States that provide communication services to the public are called **common carriers**. Their offerings and prices are described by a document called a **tariff**, which must be approved by the Federal Communications Commission for the interstate and international traffic, and by the state public utilities commissions for intrastate traffic.

At the other extreme are countries in which the national government has a complete monopoly on all communication, including the mail, telegraph, telephone, and often radio and television as well. Most of the world falls in this category. In some cases the telecommunication authority is a nationalized company, and in others it is simply a branch of the government, usually known as the **PTT** (**Post, Telegraph & Telephone** administration). Worldwide, the trend is toward liberalization and competition and away from government monopoly.

With all these different suppliers of services, there is clearly a need to provide compatibility on a worldwide scale to ensure that people (and computers) in one country can call their counterparts in another one. Actually, this need has existed for a long time. In 1865, representatives from many European governments met to form the predecessor to today's **ITU** (**International Telecommunication Union**). ITU's job was standardizing international telecommunications, which in those days meant telegraphy. Even then it was clear that if half the countries used Morse code and the other half used some other code, there was going to be a problem. When the telephone was put into international service, ITU took over the job of standardizing telephony as well. In 1947, ITU became an agency of the United Nations.

ITU has three main sectors:

1. Radiocommunications Sector (ITU-R).
2. Telecommunications Standardization Sector (ITU-T).
3. Development Sector (ITU-D).

ITU-R is concerned with allocating radio frequencies worldwide to the competing interest groups. We will be primarily concerned with ITU-T, which is concerned with telephone and data communication systems. From 1956 to 1993, ITU-T was known as **CCITT**, an acronym for its French name: Comité Consultatif International Télégraphique et Téléphonique. On March 1, 1993, CCITT was reorganized to make it less bureaucratic and renamed to reflect its new role. Both ITU-T and CCITT issued recommendations in the area of telephone and data communications. One still frequently runs into CCITT recommendations, such as CCITT X.25, although since 1993 recommendations bear the ITU-T label.

ITU-T has five classes of members:

1. Administrations (national PTTs).
2. Recognized private operators (e.g., AT&T, MCI, British Telecom).
3. Regional telecommunications organizations (e.g., the European ETSI).
4. Telecommunications vendors and scientific organizations.
5. Other interested organizations (e.g., banking and airline networks).

ITU-T has about 200 administrations, 100 private operators, and several hundred other members. Only administrations may vote, but all members may participate in ITU-T's work. Since the United States does not have a PTT, somebody else had to represent it in ITU-T. This task fell to the State Department, probably on the grounds that ITU-T had to do with foreign countries, the State Department's specialty.

ITU-T's task is to make technical recommendations about telephone, telegraph, and data communication interfaces. These often become internationally recognized standards, for example, V.24 (also known as EIA RS-232 in the United States), which specifies the placement and meaning of the various pins on the connector used by most asynchronous terminals.

It should be noted that ITU-T recommendations are technically only suggestions that governments can adopt or ignore, as they wish. In practice, a country that wishes to adopt a different telephone standard than the rest of the world is free to do so, but at the price of cutting itself off from everyone else. This might work for Albania, but elsewhere it would be a real problem. The fiction of calling ITU-T standards "recommendations" was and is necessary to keep nationalist forces in many countries placated.

The real work of ITU-T is done in Study Groups, often as large as 400 people. To make it possible to get anything at all done, the Study Groups are divided into Working Parties, which are in turn divided into Expert Teams, which are in turn divided into ad hoc groups. Once a bureaucracy, always a bureaucracy.

Despite all this, ITU-T actually gets things done. Its current output runs to about 5000 pages of recommendations a year. The members chip in to cover ITU's costs. Big, rich countries are supposed to pay up to 30 contributory units a year; small, poor ones can get away with 1/16 of a contributory unit (a contributory unit is about 250,000 dollars). It is a testimony to ITU-T's value that pretty much everyone pays their fair share, even though contributions are completely voluntary.

As telecommunications completes the transition started in the 1980s from being entirely national to being entirely global, standards will become increasingly important, and more and more organizations will want to become involved in setting them. For more information about ITU, see (Irmer, 1994).

1.7.2. Who's Who in the International Standards World

International standards are produced by **ISO** (**International Standards Organization**[†]), a voluntary, nontreaty organization founded in 1946. Its members are the national standards organizations of the 89 member countries. These members include ANSI (U.S.), BSI (Great Britain), AFNOR (France), DIN (Germany), and 85 others.

ISO issues standards on a vast number of subjects, ranging from nuts and bolts (literally) to telephone pole coatings. Over 5000 standards have been issued, including the OSI standards. ISO has almost 200 Technical Committees, numbered in the order of their creation, each dealing with a specific subject. TC1 deals with the nuts and bolts (standardizing screw thread pitches). TC97 deals with computers and information processing. Each TC has subcommittees (SCs) divided into working groups (WGs).

The real work is done largely in the WGs by over 100,000 volunteers

---

[†] For the purist, ISO's true name is the International Organization for Standardization.

worldwide. Many of these "volunteers" are assigned to work on ISO matters by their employers, whose products are being standardized. Others are government officials keen on having their country's way of doing things become the international standard. Academic experts also are active in many of the WGs.

On issues of telecommunication standards, ISO and ITU-T often cooperate (ISO is a member of ITU-T) to avoid the irony of two official and mutually incompatible international standards.

The U.S. representative in ISO is **ANSI (American National Standards Institute**), which despite its name, is a private, nongovernmental, nonprofit organization. Its members are manufacturers, common carriers, and other interested parties. ANSI standards are frequently adopted by ISO as international standards.

The procedure used by ISO for adopting standards is designed to achieve as broad a consensus as possible. The process begins when one of the national standards organizations feels the need for an international standard in some area. A working group is then formed to come up with a **CD (Committee Draft**). The CD is then circulated to all the member bodies, which get 6 months to criticize it. If a substantial majority approves, a revised document, called a **DIS (Draft International Standard**) is produced and circulated for comments and voting. Based on the results of this round, the final text of the **IS (International Standard**) is prepared, approved, and published. In areas of great controversy, a CD or DIS may have to go through several versions before acquiring enough votes, and the whole process can take years.

NIST (National Institute of Standards and Technology) is an agency of the U.S. Dept. of Commerce. It was formerly known as the National Bureau of Standards. It issues standards that are mandatory for purchases made by the U.S. Government, except for those of the Department of Defense, which has its own standards.

Another major player in the standards world is **IEEE (Institute of Electrical and Electronics Engineers**), the largest professional organization in the world. In addition to publishing scores of journals and running numerous conferences each year, IEEE has a standardization group that develops standards in the area of electrical engineering and computing. IEEE's 802 standard for local area networks is the key standard for LANs. It has subsequently been taken over by ISO as the basis for ISO 8802.

1.7.3. Who's Who in the Internet Standards World

The worldwide Internet has its own standardization mechanisms, very different from those of ITU-T and ISO. The difference can be crudely summed up by saying that the people who come to ITU or ISO standardization meetings wear suits. The people who come to Internet standardization meetings wear either jeans or military uniforms.

ITU-T and ISO meetings are populated by corporate officials and government civil servants for whom standardization is their job. They regard standardization as a good thing and devote their lives to it. Internet people, on the other hand, definitely prefer anarchy as a matter of principle, but sometimes agreement is needed to make things work. Thus standards, however regrettable, are occasionally needed.

When the ARPANET was set up, DoD created an informal committee to oversee it. In 1983, the committee was renamed the **IAB** (**Internet Activities Board**) and given a slighter broader mission, namely, to keep the researchers involved with the ARPANET and Internet pointed more-or-less in the same direction, an activity not unlike herding cats. The meaning of the acronym "IAB" was later changed to **Internet Architecture Board.**

Each of the approximately ten members of the IAB headed a task force on some issue of importance. The IAB met several times a year to discuss results and give feedback to the DoD and NSF, which were providing most of the funding at this time. When a standard was needed (e.g., a new routing algorithm), the IAB members would thrash it out and then announce the change so the graduate students who were the heart of the software effort could implement it. Communication was done by a series of technical reports called **RFCs** (**Request For Comments**). RFCs are stored on-line and can be fetched by anyone interested in them. They are numbered in chronological order of creation. Close to 2000 now exist.

By 1989, the Internet had grown so large that this highly informal style no longer worked. Many vendors by then offered TCP/IP products and did not want to change them just because ten researchers had thought of a better idea. In the summer of 1989, the IAB was reorganized again. The researchers were moved to the **IRTF** (**Internet Research Task Force**), which was made subsidiary to IAB, along with the **IETF** (**Internet Engineering Task Force**). The IAB was repopulated with people representing a broader range of organizations than just the research community. It was initially a self-perpetuating group, with members serving for a 2-year term, and new members being appointed by the old ones. Later, the **Internet Society** was created, populated by people interested in the Internet. The Internet Society is thus in a sense comparable to ACM or IEEE. It is governed by elected trustees who appoint the IAB members.

The idea of this split was to have the IRTF concentrate on long-term research, while the IETF dealt with short-term engineering issues. The IETF was divided up into working groups, each with a specific problem to solve. The chairmen of these working groups initially met together as a steering committee to direct the engineering effort. The working group topics include new applications, user information, OSI integration, routing and addressing, security, network management, and standards. Eventually, so many working groups were formed (more than 70) that they were grouped into areas, and the area chairmen met as the steering committee.

In addition, a more formal standardization process was adopted, patterned after ISOs. To become a **Proposed Standard**, the basic idea must be completely explained in an RFC and have sufficient interest in the community to warrant consideration. To advance to the **Draft Standard** stage, there must be a working implementation that has been thoroughly tested by at least two independent sites for 4 months. If the IAB is convinced that the idea is sound and the software works, it can declare the RFC to be an Internet Standard. Some Internet Standards have become DoD standards (MIL-STD), making them mandatory for DoD suppliers. David Clark once made a now-famous remark about Internet standardization consisting of "rough consensus and running code."

1.8. OUTLINE OF THE REST OF THE BOOK

This book discusses both the principles and practice of computer networking. Most chapters start with a discussion of the relevant principles, followed by a number of examples that illustrate these principles. Two networks are used as running examples throughout the text: the Internet and ATM networks. In a way, the two are complementary: ATM is mostly concerned with the lower layers, and the Internet is mostly concerned with upper layers. In the future, the Internet may run largely on an ATM backbone, so both of them may coexist. Other examples will be given where relevant.

The book is structured according to the hybrid model of Fig. 1-21. Starting with Chap. 2, we begin working our way up the protocol hierarchy beginning at the bottom. The second chapter provides some background in the field of data communication. It covers analog and digital transmission, multiplexing, switching, and the telephone system, past current, and future. This material is concerned with the physical layer, although we cover only the architectural rather than the hardware aspects. Several examples of the physical layer are also discussed, such as SONET and cellular radio.

Chap. 3 discusses the data link layer and its protocols by means of a number of increasingly complex examples. The analysis of these protocols is also covered. After that, some important real-world protocols are discussed, including HDLC (used in low- and medium-speed networks), SLIP, and PPP (used in the Internet), and ATM (used in B-ISDN).

Chap. 4 concerns the medium access sublayer, which is part of the data link layer. The basic question it deals with is how to determine who may use the network next when the network consists of a single shared channel, as in most LANs and some satellite networks. Many examples are given from the areas of LANs, fiber optic networks, and satellite networks. Bridges, which are used to connect LANs together, are also discussed here.

Chap. 5 deals with the network layer, especially routing, congestion control, and internetworking. It discusses both static and dynamic routing algorithms. Broadcast routing is also covered. The effect of poor routing, congestion, is discussed in some detail. Connecting heterogeneous networks together to form internetworks leads to numerous problems that are discussed here. The network layers in the Internet and ATM networks are given extensive coverage.

Chap. 6 deals with the transport layer. Much of the emphasis is on connection-oriented protocols, since many applications need these. An example transport service and its implementation are discussed in detail. Both the Internet transport protocols (TCP and UDP) and the ATM transport protocols (AAL 1-5) are covered in detail.

The OSI session and presentation layers are not discussed in this book as they are not widely used for anything.

Chapter 7 deals with the application layer, its protocols and applications. Among the applications covered are security, naming, electronic mail, net news, network management, the World Wide Web, and multimedia.

Chap. 8 contains an annotated list of suggested readings arranged by chapter. It is intended to help those readers who would like to pursue their study of networking further. The chapter also has an alphabetical bibliography of all references cited in this book.

1.9. SUMMARY

Computer networks can be used for numerous services, both for companies and for individuals. For companies, networks of personal computers using shared servers often provide flexibility and a good price/performance ratio. For individuals, networks offer access to a variety of information and entertainment resources.

Roughly speaking, networks can be divided up into LANs, MANs, WANs, and internetworks, each with their own characteristics, technologies, speeds, and niches. LANs cover a building, MANs cover a city, and WANs cover a country or continent. LANs and MANs are unswitched (i.e., do not have routers); WANs are switched.

Network software consists of protocols, or rules by which processes can communicate. Protocols can be either connectionless or connection-oriented. Most networks support protocol hierarchies, with each layer providing services to the layers above it and insulating them from the details of the protocols used in the lower layers. Protocol stacks are typically based either on the OSI model or the TCP/IP model. Both of these have network, transport, and application layers, but they differ on the other layers.

Well-known networks have included Novell's NetWare, the ARPANET (now defunct), NSFNET, the Internet, and various gigabit testbeds. Network services have included DQDB, SMDS, X.25, frame relay, and broadband ISDN. All of these are available commercially, from a variety of suppliers. The marketplace will determine which ones will survive and which ones will not.

PROBLEMS

1. In the future, when everyone has a home terminal connected to a computer network, instant public referendums on important pending legislation will become possible. Ultimately, existing legislatures could be eliminated, to let the will of the people be expressed directly. The positive aspects of such a direct democracy are fairly obvious; discuss some of the negative aspects.

2. An alternative to a LAN is simply a big timesharing system with terminals for all users. Give two advantages of a client-server system using a LAN.

3. A collection of five routers is to be connected in a point-to-point subnet. Between each pair of routers, the designers may put a high-speed line, a medium-speed line, a low-speed line, or no line. If it takes 100 ms of computer time to generate and inspect each topology, how long will it take to inspect all of them to find the one that best matches the expected load?

4. A group of $2^n - 1$ routers are interconnected in a centralized binary tree, with a router at each tree node. Router i communicates with router j by sending a message to the root of the tree. The root then sends the message back down to j. Derive an approximate expression for the mean number of hops per message for large n, assuming that all router pairs are equally likely.

5. A disadvantage of a broadcast subnet is the capacity wasted due to multiple hosts attempting to access the channel at the same time. As a simplistic example, suppose that time is divided into discrete slots, with each of the n hosts attempting to use the channel with probability p during each slot. What fraction of the slots are wasted due to collisions?

6. What are the SAP addresses in FM radio broadcasting?

7. What is the principal difference between connectionless communication and connection-oriented communication?

8. Two networks each provide reliable connection-oriented service. One of them offers a reliable byte stream and the other offers a reliable message stream. Are these identical? If so, why is the distinction made? If not, give an example of how they differ.

9. What is the difference between a confirmed service and an unconfirmed service? For each of the following, tell whether it might be a confirmed service, an unconfirmed service, both, or neither.
   (a) Connection establishment.
   (b) Data transmission.
   (c) Connection release.

10. What does "negotiation" mean when discussing network protocols? Give an example of it.

11. What are two reasons for using layered protocols?

12. List two ways in which the OSI reference model and the TCP/IP reference model are the same. Now list two ways in which they differ.

13. The president of the Specialty Paint Corp. gets the idea to work together with a local beer brewer for the purpose of producing an invisible beer can (as an anti-litter measure). The president tells her legal department to look into it, and they in turn ask engineering for help. As a result, the chief engineer calls his counterpart at the other company to discuss the technical aspects of the project. The engineers then report back to their respective legal departments, which then confer by telephone to arrange the legal aspects. Finally, the two corporate presidents discuss the financial side of the deal. Is this an example of a multilayer protocol in the sense of the OSI model?

14. In most networks, the data link layer handles transmission errors by requesting damaged frames to be retransmitted. If the probability of a frame's being damaged is p, what is the mean number of transmissions required to send a frame if acknowledgements are never lost?

15. Which of the OSI layers handles each of the following:
(a) Breaking the transmitted bit stream into frames.
(b) Determining which route through the subnet to use.

16. Do TPDUs encapsulate packets or the other way around? Discuss.

17. A system has an n-layer protocol hierarchy. Applications generate messages of length M bytes. At each of the layers, an h-byte header is added. What fraction of the network bandwidth is filled with headers?

18. What is the main difference between TCP and UDP?

19. Does the Novell NetWare architecture look more like X.25 or like the Internet? Explain your answer.

20. The Internet is roughly doubling in size every 18 months. Although no one really knows for sure, one estimate put the number of hosts on it at 7 million in January 1996. Use these data to compute the expected number of Internet hosts in the year 2008.

21. Why was SMDS designed as a connectionless network and frame relay as a connection-oriented one?

22. Imagine that you have trained your St. Bernard, Bernie, to carry a box of three 8mm Exabyte tapes instead of a flask of brandy. (When your disk fills up, you consider that an emergency.) These tapes each contain 7 gigabytes. The dog can travel to your side, wherever you may be, at 18 km/hour. For what range of distances does Bernie have a higher data rate than a 155-Mbps ATM line?

23. When transferring a file between two computers, (at least) two acknowledgement strategies are possible. In the first one, the file is chopped up into packets, which are individually acknowledged by the receiver, but the file transfer as a whole is not acknowledged. In the second one, the packets are not acknowledged individually, but the entire file is acknowledged when it arrives. Discuss these two approaches.

24. Imagine that the SMDS packet of Fig. 1-28 were to be incorporated in OSI protocol hierarchy. In which layer would it appear?

25. Give an advantage and a disadvantage of frame relay over a leased telephone line.

26. Why does ATM use small, fixed-length cells?

27. List two advantages and two disadvantages of having international standards for network protocols.

28. When a system has a permanent part and a removable part, such as a diskette drive and the diskette, it is important that the system be standardized, so that different companies can make both the permanent and removable parts and have everything work together. Give three examples outside the computer industry where such international standards exist. Now give three areas outside the computer industry where they do not exist.

2

THE PHYSICAL LAYER

In this chapter we will look at the lowest layer depicted in the hierarchy of Fig. 1-21. We will begin with a theoretical analysis of data transmission, only to discover that Mother (Parent?) Nature puts some limits on what can be sent over a channel.

Then we will cover transmission media, both guided (copper wire and fiber optics) and unguided (wireless). This material will provide background information on the key transmission technologies used in modern networks.

The remainder of the chapter is devoted to examples of communication systems that use these underlying transmission media. We will start with the telephone system, looking at three different versions: the current (partly) analog system, a potential digital system for the near future (N-ISDN), and a likely digital system for the distant future (ATM). Then we will look at two wireless systems, cellular radio and communication satellites.

2.1. THE THEORETICAL BASIS FOR DATA COMMUNICATION

Information can be transmitted on wires by varying some physical property such as voltage or current. By representing the value of this voltage or current as a single-valued function of time, $f(t)$, we can model the behavior of the signal and analyze it mathematically. This analysis is the subject of the following sections.

2.1.1. Fourier Analysis

In the early 19th Century, the French mathematician Jean-Baptiste Fourier proved that any reasonably behaved periodic function, $g(t)$, with period T can be constructed by summing a (possibly infinite) number of sines and cosines:

$$g(t) = \frac{1}{2}c + \sum_{n=1}^{\infty} a_n \sin(2\pi nft) + \sum_{n=1}^{\infty} b_n \cos(2\pi nft) \qquad (2\text{-}1)$$

where $f = 1/T$ is the fundamental frequency and a_n and b_n are the sine and cosine amplitudes of the nth **harmonics** (terms). Such a decomposition is called a **Fourier series**. From the Fourier series, the function can be reconstructed; that is, if the period, T, is known and the amplitudes are given, the original function of time can be found by performing the sums of Eq. (2-1).

A data signal that has a finite duration (which all of them do) can be handled by just imagining that it repeats the entire pattern over and over forever (i.e., the interval from T to $2T$ is the same as from 0 to T, etc.).

The a_n amplitudes can be computed for any given $g(t)$ by multiplying both sides of Eq. (2-1) by $\sin(2\pi kft)$ and then integrating from 0 to T. Since

$$\int_0^T \sin(2\pi kft)\sin(2\pi nft)\,dt = \begin{cases} 0 \text{ for } k \neq n \\ T/2 \text{ for } k = n \end{cases}$$

only one term of the summation survives: a_n. The b_n summation vanishes completely. Similarly, by multiplying Eq. (2-1) by $\cos(2\pi kft)$ and integrating between 0 and T, we can derive b_n. By just integrating both sides of the equation as it stands, c can be found. The results of performing these operations are as follows:

$$a_n = \frac{2}{T}\int_0^T g(t)\sin(2\pi nft)\,dt \qquad b_n = \frac{2}{T}\int_0^T g(t)\cos(2\pi nft)\,dt \qquad c = \frac{2}{T}\int_0^T g(t)\,dt$$

2.1.2. Bandwidth-Limited Signals

To see what all this has to do with data communication, let us consider a specific example: the transmission of the ASCII character "b" encoded in an 8-bit byte. The bit pattern that is to be transmitted is 01100010. The left-hand part of Fig. 2-1(a) shows the voltage output by the transmitting computer. The Fourier analysis of this signal yields the coefficients:

$$a_n = \frac{1}{\pi n}[\cos(\pi n/4) - \cos(3\pi n/4) + \cos(6\pi n/4) - \cos(7\pi n/4)]$$

$$b_n = \frac{1}{\pi n}[\sin(3\pi n/4) - \sin(\pi n/4) + \sin(7\pi n/4) - \sin(6\pi n/4)]$$

$$c = 3/8$$

The root-mean-square amplitudes, $\sqrt{a_n^2 + b_n^2}$, for the first few terms are shown on the right-hand side of Fig. 2-1(a). These values are of interest because their squares are proportional to the energy transmitted at the corresponding frequency.

No transmission facility can transmit signals without losing some power in the process. If all the Fourier components were equally diminished, the resulting signal would be reduced in amplitude but not distorted [i.e., it would have the same nice squared-off shape as Fig. 2-1(a)]. Unfortunately, all transmission facilities diminish different Fourier components by different amounts, thus introducing distortion. Usually, the amplitudes are transmitted undiminished from 0 up to some frequency f_c [measured in cycles/sec or Hertz (Hz)] with all frequencies above this cutoff frequency strongly attenuated. In some cases this is a physical property of the transmission medium, and in other cases a filter is intentionally introduced into the circuit to limit the amount of (scarce) bandwidth available to each customer.

Now let us consider how the signal of Fig. 2-1(a) would look if the bandwidth were so low that only the lowest frequencies were transmitted [i.e., the function were being approximated by the first few terms of Eq. (2-1)]. Figure 2-1(b) shows the signal that results from a channel that allows only the first harmonic (the fundamental, f) to pass through. Similarly, Fig. 2-1(c)-(e) show the spectra and reconstructed functions for higher bandwidth channels.

The time T required to transmit the character depends on both the encoding method and the signaling speed [the number of times per second that the signal changes its value (e.g., its voltage)]. The number of changes per second is measured in **baud**. A b baud line does not necessarily transmit b bits/sec, since each signal might convey several bits. If the voltages 0, 1, 2, 3, 4, 5, 6, and 7 were used, each signal value could be used to convey 3 bits, so the bit rate would be three times the baud rate. In our example, only 0s and 1s are being used as signal levels, so the bit rate is equal to the baud rate.

Given a bit rate of b bits/sec, the time required to send 8 bits (for example) is $8/b$ sec, so the frequency of the first harmonic is $b/8$ Hz. An ordinary telephone line, often called a **voice-grade line**, has an artificially introduced cutoff frequency near 3000 Hz. This restriction means that the number of the highest harmonic passed through is $3000/(b/8)$ or $24,000/b$, roughly (the cutoff is not sharp).

For some data rates, the numbers work out as shown in Fig. 2-2. From these numbers, it is clear that trying to send at 9600 bps over a voice-grade telephone line will transform Fig. 2-1(a) into something looking like Fig. 2-1(c), making accurate reception of the original binary bit stream tricky. It should be obvious that at data rates much higher than 38.4 kbps there is no hope at all for *binary* signals, even if the transmission facility is completely noiseless. In other words, limiting the bandwidth limits the data rate, even for perfect channels. However, sophisticated coding schemes that use several voltage levels do exist and can achieve higher data rates. We will discuss these later in this chapter.

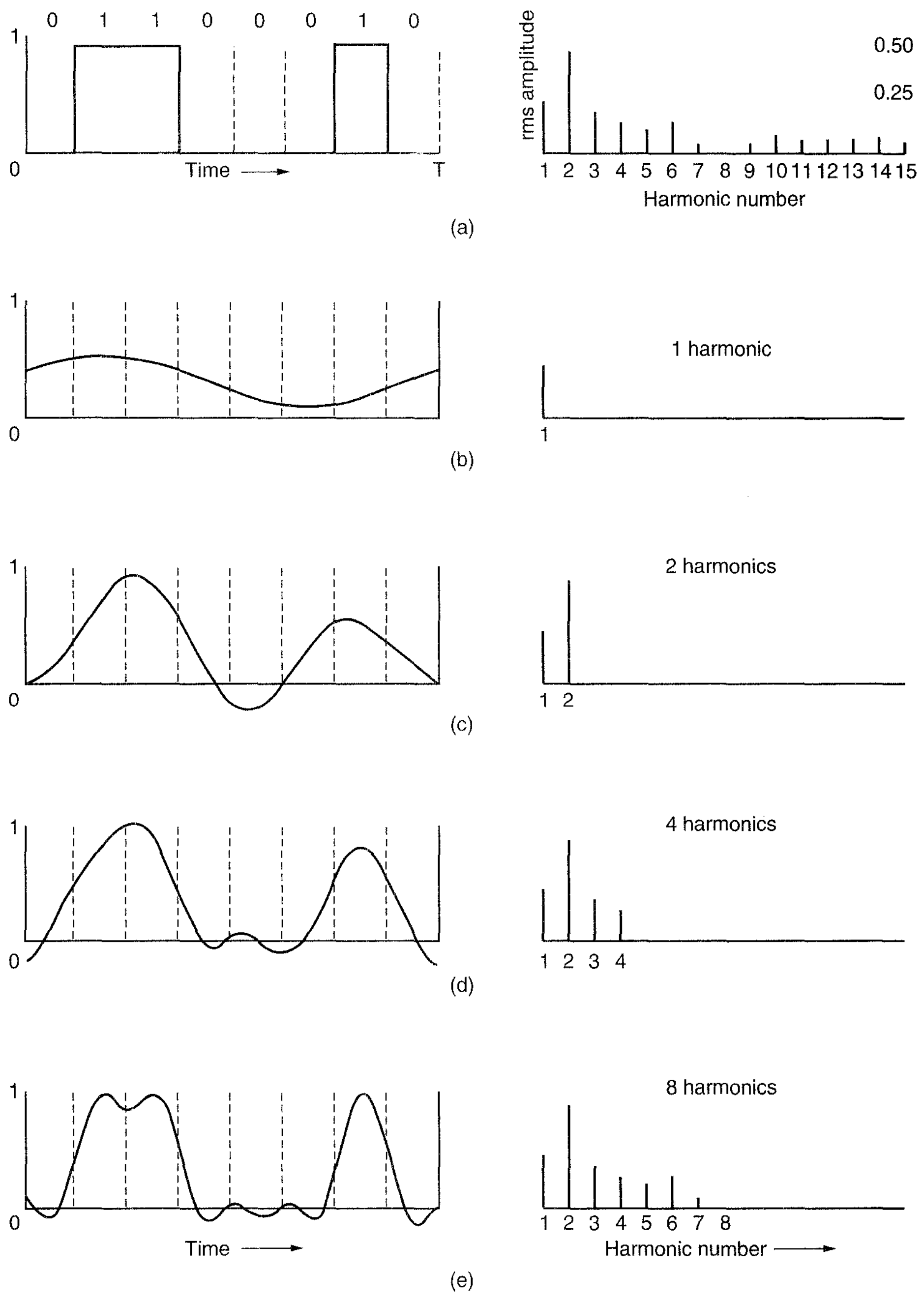


Fig. 2-1. (a) A binary signal and its root-mean-square Fourier amplitudes. (b)-(e) Successive approximations to the original signal.

Bps	T (msec)	First harmonic (Hz)	# Harmonics sent
300	26.67	37.5	80
600	13.33	75	40
1200	6.67	150	20
2400	3.33	300	10
4800	1.67	600	5
9600	0.83	1200	2
19200	0.42	2400	1
38400	0.21	4800	0

Fig. 2-2. Relation between data rate and harmonics.

2.1.3. The Maximum Data Rate of a Channel

As early as 1924, H. Nyquist realized the existence of this fundamental limit and derived an equation expressing the maximum data rate for a finite bandwidth noiseless channel. In 1948, Claude Shannon carried Nyquist's work further and extended it to the case of a channel subject to random (that is, thermodynamic) noise (Shannon, 1948). We will just briefly summarize their now classical results here.

Nyquist proved that if an arbitrary signal has been run through a low-pass filter of bandwidth H, the filtered signal can be completely reconstructed by making only $2H$ (exact) samples per second. Sampling the line faster than $2H$ times per second is pointless because the higher frequency components that such sampling could recover have already been filtered out. If the signal consists of V discrete levels, Nyquist's theorem states:

$$\text{maximum data rate} = 2H \log_2 V \text{ bits/sec}$$

For example, a noiseless 3-kHz channel cannot transmit binary (i.e., two-level) signals at a rate exceeding 6000 bps.

So far we have considered only noiseless channels. If random noise is present, the situation deteriorates rapidly. The amount of thermal noise present is measured by the ratio of the signal power to the noise power, called the **signal-to-noise ratio**. If we denote the signal power by S and the noise power by N, the signal-to-noise ratio is S/N. Usually, the ratio itself is not quoted; instead, the quantity $10 \log_{10} S/N$ is given. These units are called **decibels** (dB). An S/N ratio of 10 is 10 dB, a ratio of 100 is 20 dB, a ratio of 1000 is 30 dB and so on. The manufacturers of stereo amplifiers often characterize the bandwidth (frequency range) over which their product is linear by giving the 3-dB frequency on each end. These are the points at which the amplification factor has been approximately halved.

Shannon's major result is that the maximum data rate of a noisy channel whose bandwidth is H Hz, and whose signal-to-noise ratio is S/N, is given by

$$\text{maximum number of bits/sec} = H \log_2 (1 + S/N)$$

For example, a channel of 3000-Hz bandwidth, and a signal to thermal noise ratio of 30 dB (typical parameters of the analog part of the telephone system) can never transmit much more than 30,000 bps, no matter how many or few signal levels are used and no matter how often or how infrequent samples are taken. Shannon's result was derived using information-theory arguments and applies to any channel subject to Gaussian (thermal) noise. Counterexamples should be treated in the same category as perpetual motion machines. It should be noted, however, that this is only an upper bound and real systems rarely achieve it.

2.2. TRANSMISSION MEDIA

The purpose of the physical layer is to transport a raw bit stream from one machine to another. Various physical media can be used for the actual transmission. Each one has its own niche in terms of bandwidth, delay, cost, and ease of installation and maintenance. Media are roughly grouped into guided media, such as copper wire and fiber optics, and unguided media, such as radio and lasers through the air. We will look at these in this section and the next one.

2.2.1. Magnetic Media

One of the most common ways to transport data from one computer to another is to write them onto magnetic tape or floppy disks, physically transport the tape or disks to the destination machine, and read them back in again. While this method is not as sophisticated as using a geosynchronous communication satellite, it is often much more cost effective, especially for applications in which high bandwidth or cost per bit transported is the key factor.

A simple calculation will make this point clear. An industry standard 8-mm video tape (e.g., Exabyte) can hold 7 gigabytes. A box 50 × 50 × 50 cm can hold about 1000 of these tapes, for a total capacity of 7000 gigabytes. A box of tapes can be delivered anywhere in the United States in 24 hours by Federal Express and other companies. The effective bandwidth of this transmission is 56 gigabits/86400 sec or 648 Mbps, which is slightly better than the high-speed version of ATM (622 Mbps). If the destination is only an hour away by road, the bandwidth is increased to over 15 Gbps.

For a bank with gigabytes of data to be backed up daily on a second machine (so the bank can continue to function even in the face of a major flood or earthquake) it is likely that no other transmission technology can even begin to approach magnetic tape for performance.

If we now look at cost, we get a similar picture. The cost of 1000 video tapes is perhaps 5000 dollars when bought in bulk. A video tape can be reused at least ten times, so the tape cost is maybe 500 dollars. Add to this another 200 dollars for shipping, and we have a cost of roughly 700 dollars to ship 7000 gigabytes. This amounts to 10 cents per gigabyte. No network carrier on earth can compete with that. The moral of the story is:

> *Never underestimate the bandwidth of a station wagon full of tapes hurtling down the highway.*

2.2.2. Twisted Pair

Although the bandwidth characteristics of magnetic tape are excellent, the delay characteristics are poor. Transmission time is measured in minutes or hours, not milliseconds. For many applications an on-line connection is needed. The oldest and still most common transmission medium is **twisted pair**. A twisted pair consists of two insulated copper wires, typically about 1 mm thick. The wires are twisted together in a helical form, just like a DNA molecule. The purpose of twisting the wires is to reduce electrical interference from similar pairs close by. (Two parallel wires constitute a simple antenna; a twisted pair does not.)

The most common application of the twisted pair is the telephone system. Nearly all telephones are connected to the telephone company office by a twisted pair. Twisted pairs can run several kilometers without amplification, but for longer distances, repeaters are needed. When many twisted pairs run in parallel for a substantial distance, such as all the wires coming from an apartment building to the telephone company office, they are bundled together and encased in a protective sheath. The pairs in these bundles would interfere with one another if it were not for the twisting. In parts of the world where telephone lines run on poles above ground, it is common to see bundles several centimeters in diameter.

Twisted pairs can be used for either analog or digital transmission. The bandwidth depends on the thickness of the wire and the distance traveled, but several megabits/sec can be achieved for a few kilometers in many cases. Due to their adequate performance and low cost, twisted pairs are widely used and are likely to remain so for years to come.

Twisted pair cabling comes in several varieties, two of which are important for computer networks. **Category 3** twisted pairs consist of two insulated wires gently twisted together. Four such pairs are typically grouped together in a plastic sheath for protection and to keep the eight wires together. Prior to about 1988, most office buildings had one category 3 cable running from a central **wiring closet** on each floor into each office. This scheme allowed up to four regular telephones or two multiline telephones in each office to connect to the telephone company equipment in the wiring closet.

Starting around 1988, the more advanced **category 5** twisted pairs were introduced. They are similar to category 3 pairs, but with more twists per centimeter and Teflon insulation, which results in less crosstalk and a better quality signal over longer distances, making them more suitable for high-speed computer communication. Both of these wiring types are often referred to as **UTP (Unshielded Twisted Pair)**, to contrast them with the bulky, expensive, shielded twisted pair cables IBM introduced in the early 1980s, but which have not proven popular outside of IBM installations.

2.2.3. Baseband Coaxial Cable

Another common transmission medium is the **coaxial cable** (known to its many friends as just "coax"). It has better shielding than twisted pairs, so it can span longer distances at higher speeds. Two kinds of coaxial cable are widely used. One kind, 50-ohm cable, is commonly used for digital transmission and is the subject of this section. The other kind, 75-ohm cable, is commonly used for analog transmission and will be described in the next section. This distinction is based on historical, rather than technical, factors (e.g., early dipole antennas had an impedance of 300 ohms, and it was easy to build 4:1 impedance matching transformers).

A coaxial cable consists of a stiff copper wire as the core, surrounded by an insulating material. The insulator is encased by a cylindrical conductor, often as a closely woven braided mesh. The outer conductor is covered in a protective plastic sheath. A cutaway view of a coaxial cable is shown in Fig. 2-3.

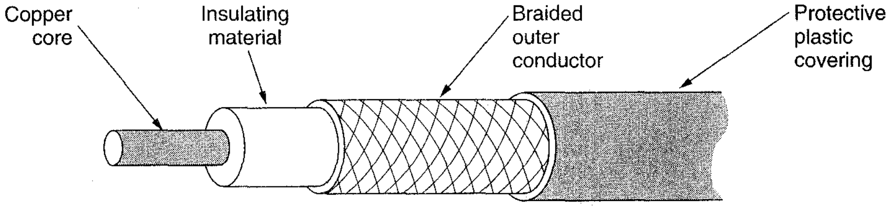


Fig. 2-3. A coaxial cable.

The construction and shielding of the coaxial cable give it a good combination of high bandwidth and excellent noise immunity. The bandwidth possible depends on the cable length. For 1-km cables, a data rate of 1 to 2 Gbps is feasible. Longer cables can also be used, but only at lower data rates or with periodic amplifiers. Coaxial cables used to be widely used within the telephone system but have now largely been replaced by fiber optics on long-haul routes. In the United States alone, 1000 km of fiber is installed every day (counting a 100-km bundle with 10 strands of fiber as 1000 km). Sprint is already 100 percent fiber, and the other major carriers are rapidly approaching that. Coax is still widely used for cable television and some local area networks, however.

2.2.4. Broadband Coaxial Cable

The other kind of coaxial cable system uses analog transmission on standard cable television cabling. It is called **broadband**. Although the term "broadband" comes from the telephone world, where it refers to anything wider than 4 kHz, in the computer networking world "broadband cable" means any cable network using analog transmission (see Cooper, 1986).

Since broadband networks use standard cable television technology, the cables can be used up to 300 MHz (and often up to 450 MHz) and can run for nearly 100 km due to the analog signaling, which is much less critical than digital signaling. To transmit digital signals on an analog network, each interface must contain electronics to convert the outgoing bit stream to an analog signal, and the incoming analog signal to a bit stream. Depending on the type of these electronics, 1 bps may occupy roughly 1 Hz of bandwidth. At higher frequencies, many bits per Hz are possible using advanced modulation techniques.

Broadband systems are divided up into multiple channels, frequently the 6-Mhz channels used for television broadcasting. Each channel can be used for analog television, CD-quality audio (1.4 Mbps), or a digital bit stream at, say, 3 Mbps, independent of the others. Television and data can be mixed on one cable.

One key difference between baseband and broadband is that broadband systems typically cover a large area and therefore need analog amplifiers to strengthen the signal periodically. These amplifiers can only transmit signals in one direction, so a computer outputting a packet will not be able to reach computers "upstream" from it if an amplifier lies between them. To get around this problem, two types of broadband systems have been developed: dual cable and single cable systems.

Dual cable systems have two identical cables running in parallel, next to each other. To transmit data, a computer outputs the data onto cable 1, which runs to a device called the **head-end** at the root of the cable tree. The head-end then transfers the signal to cable 2 for transmission back down the tree. All computers transmit on cable 1 and receive on cable 2. A dual cable system is shown in Fig. 2-4(a).

The other scheme allocates different frequency bands for inbound and outbound communication on a single cable [see Fig. 2-4(b)]. The low-frequency band is used for communication from the computers to the head-end, which then shifts the signal to the high-frequency band and rebroadcasts it. In the **subsplit** system, frequencies from 5 to 30 MHz are used for inbound traffic, and frequencies from 40 to 300 MHz are used for outbound traffic.

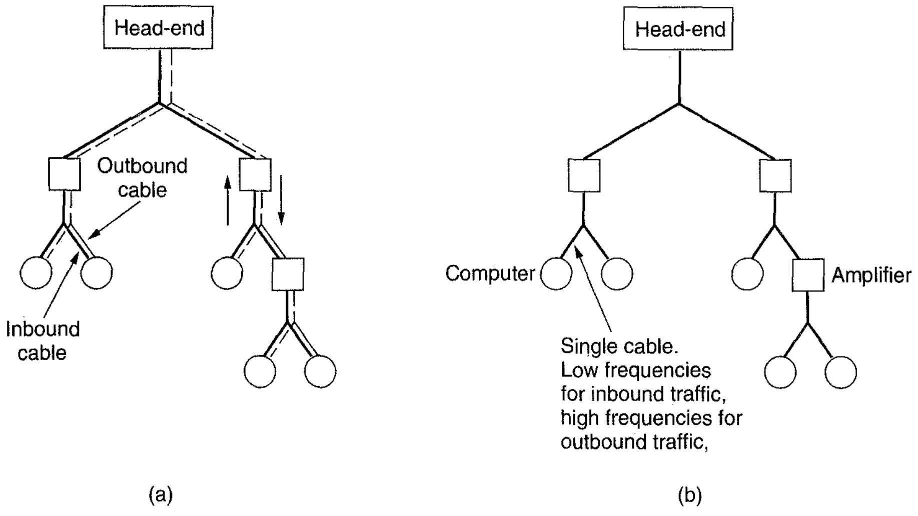


Fig. 2-4. Broadband networks. (a) Dual cable. (b) Single cable.

In the **midsplit** system, the inbound band is 5 to 116 MHz and the outbound band is 168 MHz to 300 MHz. The choice of these frequency bands is historical, having to do with how the U.S. Federal Communications Commission has assigned frequencies for television broadcasting, for which broadband was designed. Both split systems require an active head-end that accepts inbound signals on one band and rebroadcasts them on another. These techniques and frequencies were developed for cable television and have been taken over for networking without modification due to the availability of reliable and relatively inexpensive hardware.

Broadband can be used in various ways. Some computer pairs may be given a permanent channel for their exclusive use. Other computers may be able to request a channel for a temporary connection on a control channel, and then switch their frequencies to that channel for the duration of the connection. Still another arrangement is to have all the computers compete for access to a single channel or a group of channels, using techniques to be covered in Chap. 4.

Technically, broadband cable is inferior to baseband (i.e., single channel) cable for sending digital data but has the advantage that a huge amount of it is already in place. In the Netherlands, for example, 90 percent of all homes have a cable TV connection. In the United States, a TV cable runs past 80 percent of all homes. About 60 percent of these actually have a cable connection. With the competition between telephone companies and cable TV companies already in full swing, we can expect cable TV systems to begin operating as MANs and offering telephone and other services more and more often. For more information about using cable TV as a computer network, see (Karshmer and Thomas, 1992).

2.2.5. Fiber Optics

Many people in the computer industry take enormous pride in how fast computer technology is improving. In the 1970s, a fast computer (e.g., CDC 6600) could execute an instruction in 100 nsec. Twenty years later, a fast Cray computer could execute an instruction in 1 nsec, a factor of 10 improvement per decade. Not too bad.

In the same period, data communication went from 56 kbps (the ARPANET) to 1 Gbps (modern optical communication), a gain of more than a factor of 100 per decade, while at the same time the error rate went from 10^{-5} per bit to almost zero.

Furthermore, single CPUs are beginning to approach physical limits, such as speed of light and heat dissipation problems. In contrast, with *current* fiber technology, the achievable bandwidth is certainly in excess of 50,000 Gbps (50 Tbps) and many people are looking very hard for better materials. The current practical signaling limit of about 1 Gbps is due to our inability to convert between electrical and optical signals any faster. In the laboratory, 100 Gbps is feasible on short runs. A speed of 1 terabit/sec is only a few years down the road. Fully optical systems, including getting into and out of the computer, are within reach (Miki, 1994a).

In the race between computing and communication, communication won. The full implications of essentially infinite bandwidth (although not at zero cost) have not yet sunk in to a generation of computer scientists and engineers taught to think in terms of the low Nyquist and Shannon limits imposed by copper wire. The new conventional wisdom should be that all computers are hopelessly slow, and networks should try to avoid computation at all costs, no matter how much bandwidth that wastes. In this section we will study fiber optics to see how that transmission technology works.

An optical transmission system has three components: the light source, the transmission medium, and the detector. Conventionally, a pulse of light indicates a 1 bit and the absence of light indicates a zero bit. The transmission medium is an ultra-thin fiber of glass. The detector generates an electrical pulse when light falls on it. By attaching a light source to one end of an optical fiber and a detector to the other, we have a unidirectional data transmission system that accepts an electrical signal, converts and transmits it by light pulses, and then reconverts the output to an electrical signal at the receiving end.

This transmission system would leak light and be useless in practice except for an interesting principle of physics. When a light ray passes from one medium to another, for example, from fused silica to air, the ray is refracted (bent) at the silica/air boundary as shown in Fig. 2-5. Here we see a light ray incident on the boundary at an angle α_1 emerging at an angle β_1. The amount of refraction depends on the properties of the two media (in particular, their indices of refraction). For angles of incidence above a certain critical value, the light is refracted back into the silica; none of it escapes into the air. Thus a light ray incident at or above the critical angle is trapped inside the fiber, as shown in Fig. 2-5(b), and can propagate for many kilometers with virtually no loss.

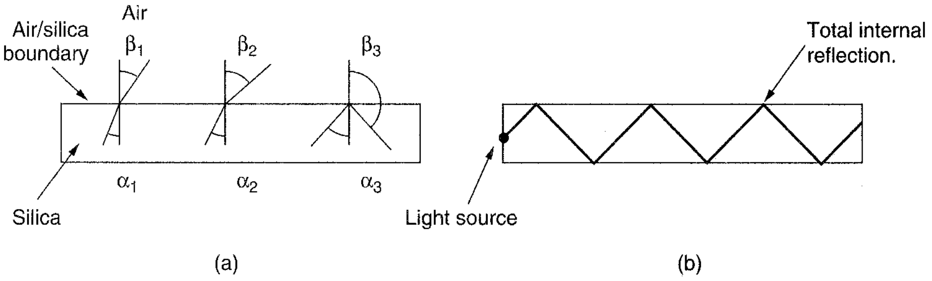


Fig. 2-5. (a) Three examples of a light ray from inside a silica fiber impinging on the air/silica boundary at different angles. (b) Light trapped by total internal reflection.

The sketch of Fig. 2-5(b) shows only one trapped ray, but since any light ray incident on the boundary above the critical angle will be reflected internally, many different rays will be bouncing around at different angles. Each ray is said to have a different **mode** so a fiber having this property is called a **multimode fiber**.

However, if the fiber's diameter is reduced to a few wavelengths of light, the fiber acts like a wave guide, and the light can only propagate in a straight line, without bouncing, yielding a **single-mode fiber**. Single mode fibers are more expensive but can be used for longer distances. Currently available single-mode fibers can transmit data at several Gbps for 30 km. Even higher data rates have been achieved in the laboratory for shorter distances. Experiments have shown that powerful lasers can drive a fiber 100 km long without repeaters, although at lower speeds. Research on erbium-doped fibers promises even longer runs without repeaters.

Transmission of Light through Fiber

Optical fibers are made of glass, which, in turn, is made from sand, an inexpensive raw material available in unlimited amounts. Glass making was known to the ancient Egyptians, but their glass had to be no more than 1 mm thick or the light could not shine through. Glass transparent enough to be useful for windows was developed during the Renaissance. The glass used for modern optical fibers is so transparent that if the oceans were full of it instead of water, the seabed would as visible from the surface as the ground is from an airplane on a clear day.

The attenuation of light through glass depends on the wavelength of the light.

For the kind of glass used in fibers, the attenuation is shown in Fig. 2-6 in decibels per linear kilometer of fiber. The attenuation in decibels is given by the formula

$$\text{Attenuation in decibels} = 10 \log_{10} \frac{\text{transmitted power}}{\text{received power}}$$

For example, a factor of two loss gives an attenuation of $10 \log_{10} 2 = 3$ dB. The figure shows the near infrared part of the spectrum, which is what is used in practice. Visible light has slightly shorter wavelengths, from 0.4 to 0.7 microns (1 micron is 10^{-6} meters).

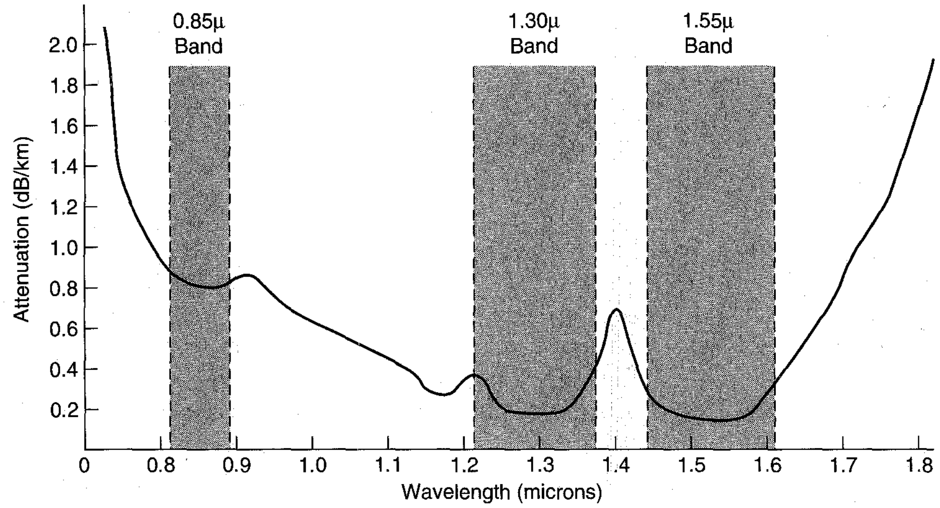


Fig. 2-6. Attenuation of light through fiber in the infrared region.

Three wavelength bands are used for communication. They are centered at 0.85, 1.30, and 1.55 microns, respectively. The latter two have good attenuation properties (less than 5 percent loss per kilometer). The 0.85 micron band has higher attenuation, but the nice property that at that wavelength, the lasers and electronics can be made from the same material (gallium arsenide). All three bands are 25,000 to 30,000 GHz wide.

Light pulses sent down a fiber spread out in length as they propagate. This spreading is called **dispersion**. The amount of it is wavelength dependent. One way to keep these spread-out pulses from overlapping is to increase the distance between them, but this can only be done by reducing the signaling rate. Fortunately, it has been discovered that by making the pulses in a special shape related to the reciprocal of the hyperbolic cosine, all the dispersion effects cancel out, and it may be possible to send pulses for thousands of kilometers without appreciable shape distortion. These pulses are called **solitons**. A considerable amount of research is going on to take solitons out of the lab and into the field.

Fiber Cables

Fiber optic cables are similar to coax, except without the braid. Figure 2-7(a) shows a single fiber viewed from the side. At the center is the glass core through which the light propagates. In multimode fibers, the core is 50 microns in diameter, about the thickness of a human hair. In single-mode fibers the core is 8 to 10 microns.

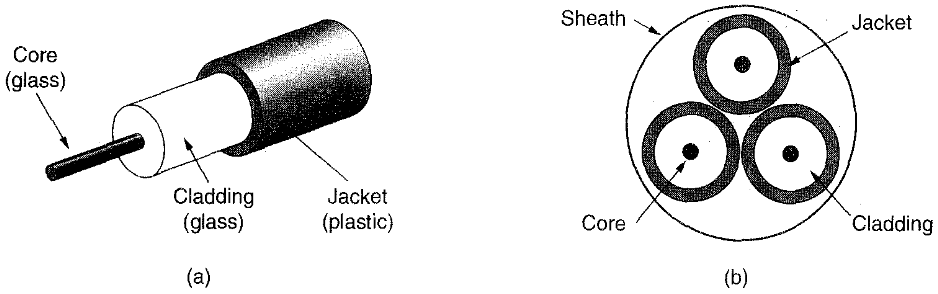


Fig. 2-7. (a) Side view of a single fiber. (b) End view of a sheath with three fibers.

The core is surrounded by a glass cladding with a lower index of refraction than the core, to keep all the light in the core. Next comes a thin plastic jacket to protect the cladding. Fibers are typically grouped together in bundles, protected by an outer sheath. Figure 2-7(b) shows a sheath with three fibers.

Terrestrial fiber sheaths are normally laid in the ground within a meter of the surface, where they are occasionally subject to attacks by backhoes or gophers. Near the shore, transoceanic fiber sheaths are buried in trenches by a kind of seaplow. In deep water, they just lie on the bottom, where they can be snagged by fishing trawlers or eaten by sharks.

Fibers can be connected in three different ways. First, they can terminate in connectors and be plugged into fiber sockets. Connectors lose about 10 to 20 percent of the light, but they make it easy to reconfigure systems.

Second, they can be spliced mechanically. Mechanical splices just lay the two carefully cut ends next to each other in a special sleeve and clamp them in place. Alignment can be improved by passing light through the junction and then making small adjustments to maximize the signal. Mechanical splices take trained personnel about 5 minutes, and result in a 10 percent light loss.

Third, two pieces of fiber can be fused (melted) to form a solid connection. A fusion splice is almost as good as a single drawn fiber, but even here, a small amount of attenuation occurs. For all three kinds of splices, reflections can occur at the point of the splice, and the reflected energy can interfere with the signal.

Two kinds of light sources can be used to do the signaling, LEDs (Light Emitting Diodes) and semiconductor lasers. They have different properties, as shown in Fig. 2-8. They can be tuned in wavelength by inserting Fabry-Perot or Mach-Zehnder interferometers between the source and the fiber. Fabry-Perot interferometers are simple resonant cavities consisting of two parallel mirrors. The light is incident perpendicularly to the mirrors. The length of the cavity selects out those wavelengths that fit inside an integral number of times. Mach-Zehnder interferometers separate the light into two beams. The two beams travel slightly different distances. They are recombined at the end and are in phase for only certain wavelengths.

Item	LED	Semiconductor laser
Data rate	Low	High
Mode	Multimode	Multimode or single mode
Distance	Short	Long
Lifetime	Long life	Short life
Temperature sensitivity	Minor	Substantial
Cost	Low cost	Expensive

Fig. 2-8. A comparison of semiconductor diodes and LEDs as light sources.

The receiving end of an optical fiber consists of a photodiode, which gives off an electrical pulse when struck by light. The typical response time of a photodiode is 1 nsec, which limits data rates to about 1 Gbps. Thermal noise is also an issue, so a pulse of light must carry enough energy to be detected. By making the pulses powerful enough, the error rate can be made arbitrarily small.

Fiber Optic Networks

Fiber optics can be used for LANs as well as for long-haul transmission, although tapping onto it is more complex than connecting to an Ethernet. One way around the problem is to realize that a ring network is really just a collection of point-to-point links, as shown in Fig. 2-9. The interface at each computer passes the light pulse stream through to the next link and also serves as a T junction to allow the computer to send and accept messages.

Two types of interfaces are used. A passive interface consists of two taps fused onto the main fiber. One tap has an LED or laser diode at the end of it (for transmitting), and the other has a photodiode (for receiving). The tap itself is completely passive and is thus extremely reliable because a broken LED or photodiode does not break the ring. It just takes one computer off-line.

The other interface type, shown in Fig. 2-9, is the **active repeater**. The incoming light is converted to an electrical signal, regenerated to full strength if it

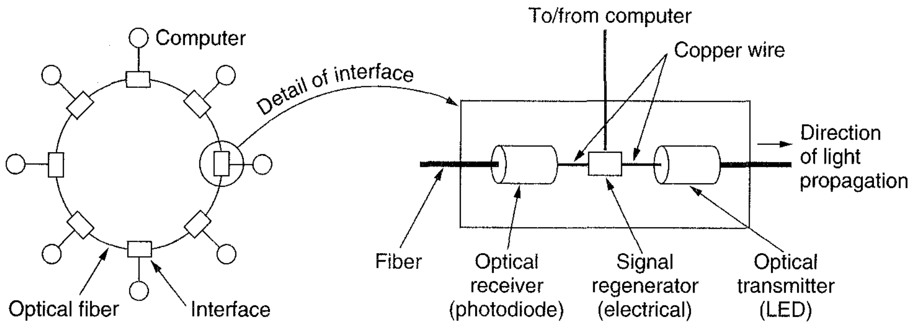


Fig. 2-9. A fiber optic ring with active repeaters.

has been weakened, and retransmitted as light. The interface with the computer is an ordinary copper wire that comes into the signal regenerator. Purely optical repeaters are now being used, too. These devices do not require the optical to electrical to optical conversions, which means they can operate at extremely high bandwidths.

If an active repeater fails, the ring is broken and the network goes down. On the other hand, since the signal is regenerated at each interface, the individual computer-to-computer links can be kilometers long, with virtually no limit on the total size of the ring. The passive interfaces lose light at each junction, so the number of computers and total ring length are greatly restricted.

A ring topology is not the only way to build a LAN using fiber optics. It is also possible to have hardware broadcasting using the **passive star** construction of Fig. 2-10. In this design, each interface has a fiber running from its transmitter to a silica cylinder, with the incoming fibers fused to one end of the cylinder. Similarly, fibers fused to the other end of the cylinder are run to each of the receivers. Whenever an interface emits a light pulse, it is diffused inside the passive star to illuminate all the receivers, thus achieving broadcast. In effect, the passive star combines all the incoming signals and transmits the merged result out on all lines. Since the incoming energy is divided among all the outgoing lines, the number of nodes in the network is limited by the sensitivity of the photodiodes.

Comparison of Fiber Optics and Copper Wire

It is instructive to compare fiber to copper. Fiber has many advantages. To start with, it can handle much higher bandwidths than copper. This alone would require its use in high-end networks. Due to the low attenuation, repeaters are needed only about every 30 km on long lines, versus about every 5 km for copper, a substantial cost saving. Fiber also has the advantage of not being affected by

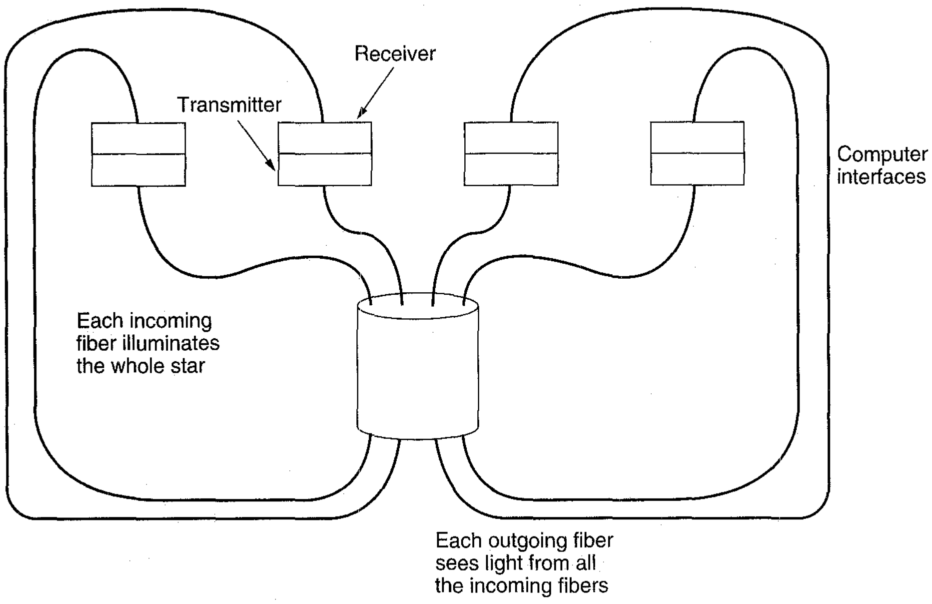


Fig. 2-10. A passive star connection in a fiber optics network.

power surges, electromagnetic interference, or power failures. Nor is it affected by corrosive chemicals in the air, making it ideal for harsh factory environments.

Oddly enough, telephone companies like fiber for a different reason: it is thin and lightweight. Many existing cable ducts are completely full, so there is no room to add new capacity. Removing all the copper and replacing it by fibers empties up the ducts, and the copper has excellent resale value to copper refiners who see it as very high grade ore. Also fiber is lighter than copper. One thousand twisted pairs 1 km long weigh 8000 kg. Two fibers have more capacity and weigh only 100 kg, which greatly reduces the need for expensive mechanical support systems that must be maintained. For new routes, fiber wins hands down due to its much lower installation cost.

Finally, fibers do not leak light and are quite difficult to tap. This gives them excellent security against potential wiretappers.

The reason that fiber is better than copper is inherent in the underlying physics. When electrons move in a wire, they affect one another and are themselves affected by electrons outside the wire. Photons in a fiber do not affect one another (they have no electric charge) and are not affected by stray photons outside the fiber.

On the downside, fiber is an unfamiliar technology requiring skills most engineers do not have. Since optical transmission is inherently unidirectional, two-way communication requires either two fibers or two frequency bands on one fiber. Finally, fiber interfaces cost more than electrical interfaces. Nevertheless, the future of all fixed data communication for distances of more than a few meters is clearly with fiber. For a detailed discussion of all aspects of fiber optic networks, see (Green, 1993).

2.3. WIRELESS TRANSMISSION

Our age has given rise to information junkies: people who need to be on-line all the time. For these mobile users, twisted pair, coax, and fiber optics are of no use. They need to get their hits of data for their laptop, notebook, shirt pocket, palmtop, or wristwatch computers without being tethered to the terrestrial communication infrastructure. For these users, wireless communication is the answer. In this section we will look at wireless communication in general, as it has many other important applications besides providing connectivity to users who want to read their email in airplanes.

Some people even believe that the future holds only two kinds of communication: fiber and wireless. All fixed (i.e., nonmobile) computers, telephones, faxes, and so on will be by fiber, and all mobile ones will use wireless.

However wireless also has advantages for even fixed devices in some circumstances. For example, if running a fiber to a building is difficult due to the terrain (mountains, jungles, swamps, etc.) wireless may be preferable. It is noteworthy that modern wireless digital communication began in the Hawaiian Islands, where large chunks of Pacific Ocean separated the users and the telephone system was inadequate.

2.3.1. The Electromagnetic Spectrum

When electrons move, they create electromagnetic waves that can propagate through free space (even in a vacuum). These waves were predicted by the British physicist James Clerk Maxwell in 1865 and first produced and observed by the German physicist Heinrich Hertz in 1887. The number of oscillations per second of an electromagnetic wave is called its **frequency**, f, and is measured in **Hz** (in honor of Heinrich Hertz). The distance between two consecutive maxima (or minima) is called the **wavelength**, which is universally designated by the Greek letter λ (lambda).

By attaching an antenna of the appropriate size to an electrical circuit, the electromagnetic waves can be broadcast efficiently and received by a receiver some distance away. All wireless communication is based on this principle.

In vacuum, all electromagnetic waves travel at the same speed, no matter what their frequency. This speed, usually called the **speed of light**, c, is approximately 3×10^8 m/sec, or about 1 foot (30 cm) per nanosecond. In copper or fiber the speed slows to about 2/3 of this value and becomes slightly frequency dependent. The speed of light is the ultimate speed limit. No object or signal can ever move faster than it.

The fundamental relation between f, λ, and c (in vacuum) is

$$\lambda f = c \tag{2-2}$$

Since c is a constant, if we know f we can find λ and vice versa. For example, 1-MHz waves are about 300 meters long and 1-cm waves have a frequency of 30 GHz.

The electromagnetic spectrum is shown in Fig. 2-11. The radio, microwave, infrared, and visible light portions of the spectrum can all be used for transmitting information by modulating the amplitude, frequency, or phase of the waves. Ultraviolet light, X-rays, and gamma rays would be even better, due to their higher frequencies, but they are hard to produce and modulate, do not propagate well through buildings, and are dangerous to living things. The bands listed at the bottom of Fig. 2-11 are the official ITU names and are based on the wavelengths, so the LF band goes from 1 km to 10 km (approximately 30 kHz to 300 kHz). The terms LF, MF, and HF refer to low, medium, and high frequency, respectively. Clearly, when the names were assigned, nobody expected to go above 10 MHz, so the higher bands were later named the Very, Ultra, Super, Extremely, and Tremendously High Frequency bands. Beyond that there are no names, but Incredibly, Astonishingly, and Prodigiously high frequency (IHF, AHF, and PHF) would sound nice.

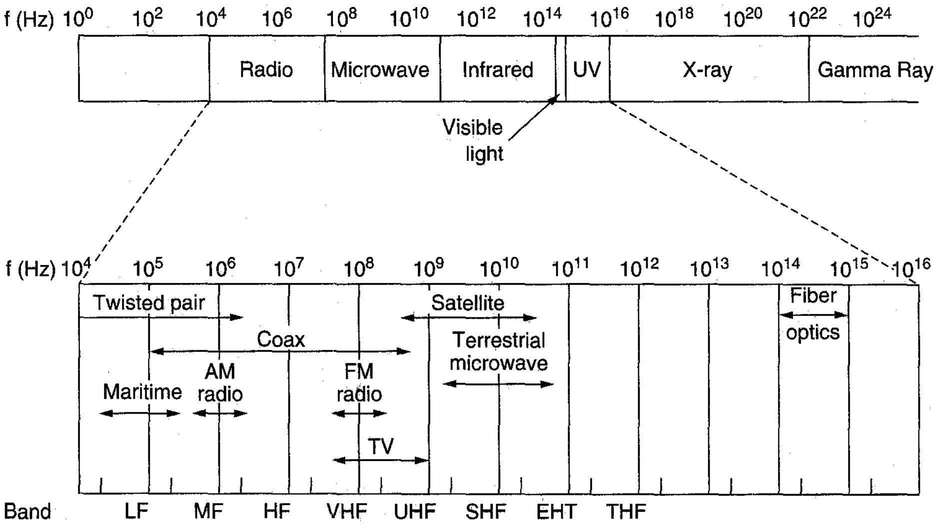


Fig. 2-11. The electromagnetic spectrum and its uses for communication.

The amount of information that an electromagnetic wave can carry is related to its bandwidth. With current technology, it is possible to encode a few bits per Hertz at low frequencies, but often as many as 40 under certain conditions at high frequencies, so a cable with a 500 MHz bandwidth can carry several gigabits/sec. From Fig. 2-11 it should now be obvious why networking people like fiber optics so much.

If we solve Eq. (2-2) for f and differentiate with respect to λ we get

$$\frac{df}{d\lambda} = -\frac{c}{\lambda^2}$$

If we now go to finite differences instead of differentials and only look at absolute values, we get

$$\Delta f = \frac{c\Delta\lambda}{\lambda^2} \tag{2-3}$$

Thus given the width of a wavelength band, $\Delta\lambda$, we can compute the corresponding frequency band, Δf, and from that the data rate the band can produce. The wider the band, the higher the data rate. As an example, consider the 1.30-micron band of Fig. 2-6. Here we have $\lambda = 1.3 \times 10^{-6}$ and $\Delta\lambda = 0.17 \times 10^{-6}$, so Δf is about 30 THz.

To prevent total chaos, there are national and international agreements about who gets to use which frequencies. Since everyone wants a higher data rate, everyone wants more spectrum. In the United States, the FCC allocates spectrum for AM and FM radio, television, and cellular phones, as well as for telephone companies, police, maritime, navigation, military, government, and many other competing users. Worldwide, an agency of ITU-R (WARC) does this work. In the meeting in Spain in 1991, for example, WARC allocated some spectrum to hand-held personal communicators. Unfortunately, the FCC, which is not bound by WARC's recommendations, chose a different piece (because the people in the United States who had the band WARC chose did not want to give it up and had enough political clout to prevent that). Consequently, personal communicators built for the U.S. market will not work in Europe or Asia, and vice versa.

Most transmissions use a narrow frequency band (i.e., $\Delta f / f \ll 1$) to get the best reception (many watts/Hz). However, in some cases, the transmitter hops from frequency to frequency in a regular pattern or the transmissions are intentionally spread out over a wide frequency band. This technique is called **spread spectrum** (Kohno et al., 1995). It is popular for military communication because it makes transmissions hard to detect and next to impossible to jam. Frequency hopping is not of much interest to us (other than to note that it was co-invented by the movie actress Hedy Lamarr). True spread spectrum, sometimes called **direct sequence spread spectrum**, is gaining popularity in the commercial world, and we will come back to it in Chap. 4. For a fascinating and detailed history of spread spectrum communication, see (Scholtz, 1982).

For the moment, we will assume that all transmissions use a narrow frequency band. We will now discuss how the various parts of the spectrum are used, starting with radio.

2.3.2. Radio Transmission

Radio waves are easy to generate, can travel long distances, and penetrate buildings easily, so they are widely used for communication, both indoors and outdoors. Radio waves also are omnidirectional, meaning that they travel in all directions from the source, so that the transmitter and receiver do not have to be carefully aligned physically.

Sometimes omnidirectional radio is good, but sometimes it is bad. In the 1970s, General Motors decided to equip its new Cadillacs with computer-controlled antilock brakes. When the driver stepped on the brake pedal, the computer pulsed the brakes on and off instead of locking them on hard. One fine day an Ohio Highway Patrolman began using his new mobile radio to call headquarters, and suddenly the Cadillac next to him began behaving like a bucking bronco. When the officer pulled the car over, the driver claimed that he had done nothing and that the car had gone crazy.

Eventually, a pattern began to emerge: Cadillacs would sometimes go berserk, but only on major highways in Ohio and then only when the Highway Patrol was watching. For a long, long time General Motors could not understand why Cadillacs worked fine in all the other states, and also on minor roads in Ohio. Only after a considerable amount of searching did they discover that the Cadillac's wiring made a fine antenna for the frequency the Ohio Highway Patrol's new radio system used.

The properties of radio waves are frequency dependent. At low frequencies, radio waves pass through obstacles well, but the power falls off sharply with distance from the source, roughly as $1/r^3$ in air. At high frequencies, radio waves tend to travel in straight lines and bounce off obstacles. They are also absorbed by rain. At all frequencies, radio waves are subject to interference from motors and other electrical equipment.

Due to radio's ability to travel long distances, interference between users is a problem. For this reason, all governments tightly license the user of radio transmitters, with one exception (discussed below).

In the VLF, LF, and MF bands, radio waves follow the ground, as illustrated in Fig. 2-12(a). These waves can be detected for perhaps 1000 km at the lower frequencies, less at the higher ones. AM radio broadcasting uses the MF band, which is why Boston AM radio stations cannot be heard easily in New York. Radio waves in these bands easily pass through buildings, which is why portable radios work indoors. The main problem with using these bands for data communication is the relative low bandwidth they offer [see Eq. (2-2)].

In the HF and VHF bands, the ground waves tend to be absorbed by the earth.

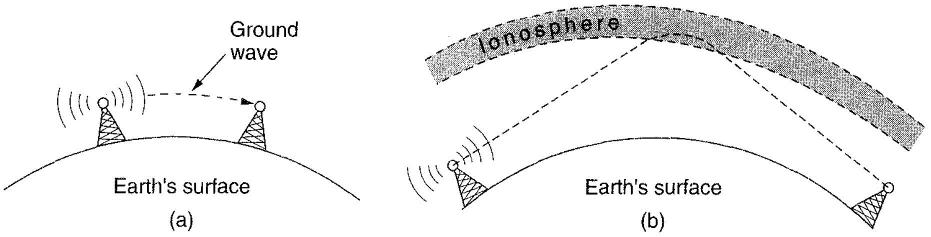


Fig. 2-12. (a) In the VLF, VF, and MF bands, radio waves follow the curvature of the earth. (b) In the HF they bounce off the ionosphere.

However, the waves that reach the ionosphere, a layer of charged particles circling the earth at a height of 100 to 500 km, are refracted by it and sent back to earth, as shown in Fig. 2-12(b). Under certain atmospheric conditions, the signals may bounce several times. Amateur radio operators (hams) use these bands to talk long distance. The military also communicates in the HF and VHF bands.

2.3.3. Microwave Transmission

Above 100 MHz, the waves travel in straight lines and can therefore be narrowly focused. Concentrating all the energy into a small beam using a parabolic antenna (like the familiar satellite TV dish) gives a much higher signal to noise ratio, but the transmitting and receiving antennas must be accurately aligned with each other. In addition, this directionality allows multiple transmitters lined up in a row to communicate with multiple receivers in a row without interference. Before fiber optics, for decades these microwaves formed the heart of the long-distance telephone transmission system. In fact, the long-distance carrier MCI's name first stood for Microwave Communications, Inc., because its entire system was originally built on microwave towers (it has since upgraded major portions of its network to fiber).

Since the microwaves travel in a straight line, if the towers are too far apart, the earth will get in the way (think about a San Francisco to Amsterdam link). Consequently, repeaters are needed periodically. The higher the towers are, the further apart they can be. The distance between repeaters goes up very roughly with the square root of the tower height. For 100-m high towers, repeaters can be spaced 80 km apart.

Unlike radio waves at lower frequencies, microwaves do not pass through buildings well. In addition, even though the beam may be well focused at the transmitter, there is still some divergence in space. Some waves may be refracted off low-lying atmospheric layers and may take slightly longer to arrive than direct waves. The delayed waves may arrive out of phase with the direct wave and thus cancel the signal. This effect is called **multipath fading** and is often a serious problem. It is weather and frequency dependent. Some operators keep 10 percent of their channels idle as spares to switch on when multipath fading wipes out some frequency band temporarily.

The demand for more and more spectrum works to keep improving the technology so transmissions can use still higher frequencies. Bands up to 10 GHz are now in routine use, but at about 8 GHz a new problem sets in: absorption by water. These waves are only a few centimeters long and are absorbed by rain. This effect would be fine if one were planning to build a huge outdoor microwave oven, but for communication, it is a severe problem. As with multipath fading, the only solution is to shut off links that are being rained on and route around them.

In summary, microwave communication is so widely used for long-distance telephone communication, cellular telephones, television distribution, and other uses, that a severe shortage of spectrum has developed. It has several significant advantages over fiber. The main one is that no right of way is needed, and by buying a small plot of ground every 50 km and putting a microwave tower on it, one can bypass the telephone system and communicate directly. This is how MCI managed to get started as a new long-distance telephone company so quickly. (Sprint went a different route: it was formed by the Southern Pacific Railroad, which already owned a large amount of right of way, and just buried fiber next to the tracks.)

Microwave is also relatively inexpensive. Putting up two simple towers (maybe just big poles with four guy wires) and putting antennas on each one may be cheaper than burying 50 km of fiber through a congested urban area or up over a mountain, and it may also be cheaper than leasing the telephone company's fiber, especially if the telephone company has not yet even fully paid for the copper it ripped out when it put in the fiber.

In addition to being used for long-distance transmission, microwaves have another important use, namely, the **Industrial/Scientific/Medical** bands. These bands form the one exception to the licensing rule: transmitters using these bands do not require government licensing. One band is allocated worldwide: 2.400–2.484 GHz. In addition, in the United States and Canada, bands also exist from 902–928 MHz and from 5.725–5.850 GHz. These bands are used for cordless telephones, garage door openers, wireless hi-fi speakers, security gates, etc. The 900-MHz band works best but is crowded and equipment using it may only be operated in North America. The higher bands require more expensive electronics and are subject to interference from microwave ovens and radar installations. Nevertheless, these bands are popular for various forms of short-range wireless networking because they avoid the problems associated with licensing.

2.3.4. Infrared and Millimeter Waves

Unguided infrared and millimeter waves are widely used for short-range communication. The remote controls used on televisions, VCRs, and stereos all use infrared communication. They are relatively directional, cheap, and easy to build, but have a major drawback: they do not pass through solid objects (try standing between your remote control and your television and see if it still works). In general, as we go from long-wave radio toward visible light, the waves behave more and more like light and less and less like radio.

On the other hand, the fact that infrared waves do not pass through solid walls well is also a plus. It means that an infrared system in one room of a building will not interfere with a similar system in adjacent rooms. Furthermore, security of infrared systems against eavesdropping is better than that of radio systems precisely for this reason. For these reasons, no government license is needed to operate an infrared system, in contrast to radio systems, which must be licensed.

These properties have made infrared an interesting candidate for indoor wireless LANs. For example, the computers and offices in a building can be equipped with relatively unfocused (i.e., somewhat omnidirectional) infrared transmitters and receivers. In this way, portable computers with infrared capability can be on the local LAN without having to physically connect to it. When several people show up for a meeting with their portables, they can just sit down in the conference room and be fully connected, without having to plug in. Infrared communication cannot be used outdoors because the sun shines as brightly in the infrared as in the visible spectrum. For more information about infrared communication, see (Adams et al., 1993; and Bantz and Bauchot, 1994).

2.3.5. Lightwave Transmission

Unguided optical signaling has been in use for centuries. Paul Revere used binary optical signaling from the Old North Church just prior to his famous ride. A more modern application is to connect the LANs in two buildings via lasers mounted on their rooftops. Coherent optical signaling using lasers is inherently unidirectional, so each building needs its own laser and its own photodetector. This scheme offers very high bandwidth and very low cost. It is also relatively easy to install, and, unlike microwave, does not require an FCC license.

The laser's strength, a very narrow beam, is also its weakness here. Aiming a laser beam 1 mm wide at a target 1 mm wide 500 meters away requires the marksmanship of a latter-day Annie Oakley. Usually, lenses are put into the system to defocus the beam slightly.

A disadvantage is that laser beams cannot penetrate rain or thick fog, but they normally work well on sunny days. However, the author once attended a conference at a modern hotel in Europe at which the conference organizers thoughtfully provided a room full of terminals for the attendees to read their email during boring presentations. Since the local PTT was unwilling to install a large number of telephone lines for just 3 days, the organizers put a laser on the roof and aimed it at their university's computer science building a few kilometers away. They tested it the night before the conference and it worked perfectly. At 9 a.m. the next morning, on a bright sunny day, the link failed completely and stayed down all day. That evening, the organizers tested it again very carefully, and once again it worked absolutely perfectly. The pattern repeated itself for two more days consistently.

After the conference, the organizers discovered the problem. Heat from the sun during the daytime caused convection currents to rise up from the roof of the building, as shown in Fig. 2-13. This turbulent air diverted the beam and made it dance around the detector. Atmospheric "seeing" like this makes the stars twinkle (which is why astronomers put their telescopes on the tops of mountains—to get above as much of the atmosphere as possible). It is also responsible for shimmering roads on a hot day and the wavy images when looking out above a hot radiator.

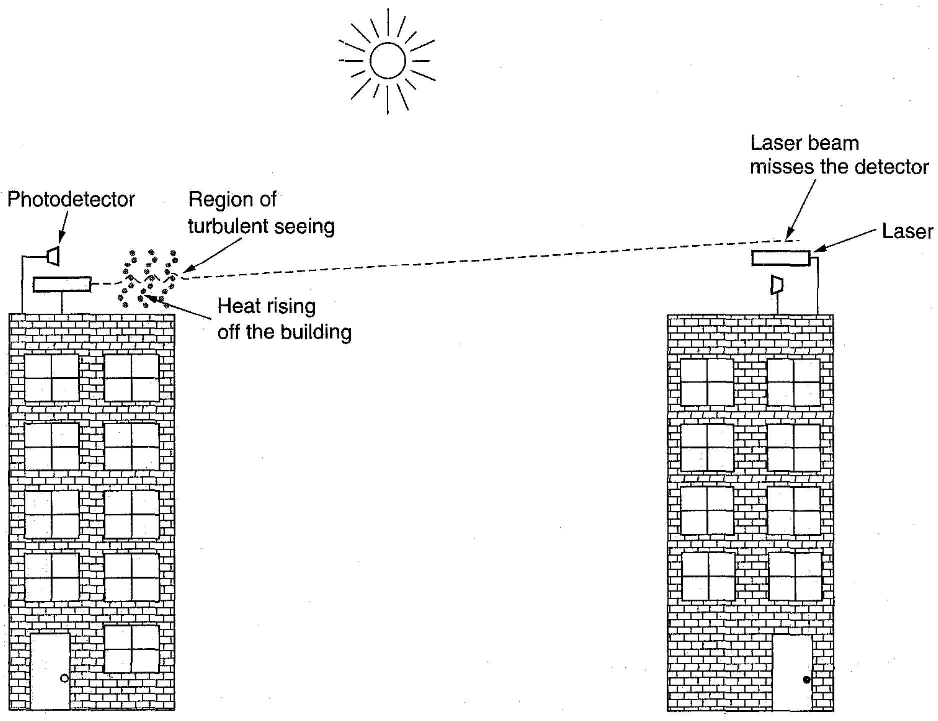


Fig. 2-13. Convection currents can interfere with laser communication systems. A bidirectional system, with two lasers, is pictured here.

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.

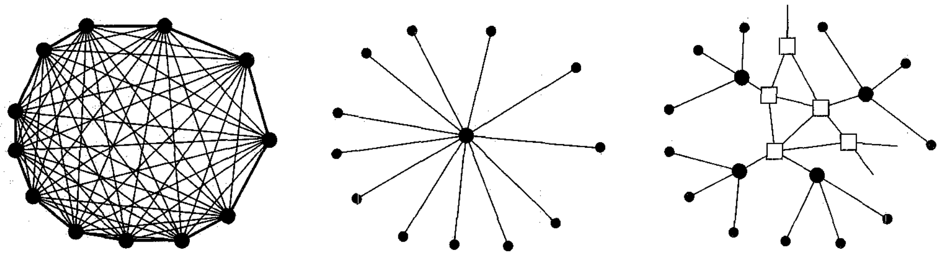
To see the order of magnitude of the problem, let us make a rough but illustrative comparison of the properties of a typical computer-computer connection via a local cable and via a dial-up telephone line. A cable running between two computers can transfer data at memory speeds, typically 10^7 to 10^8 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.



(a) (b) (c)

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.

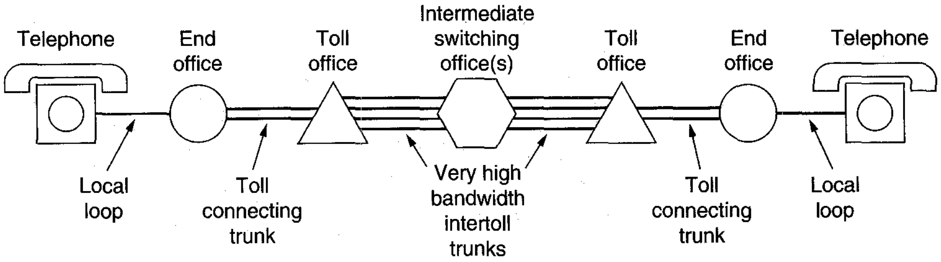


Fig. 2-15. Typical circuit route for a medium-distance call.

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** (**IntereXchange 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).

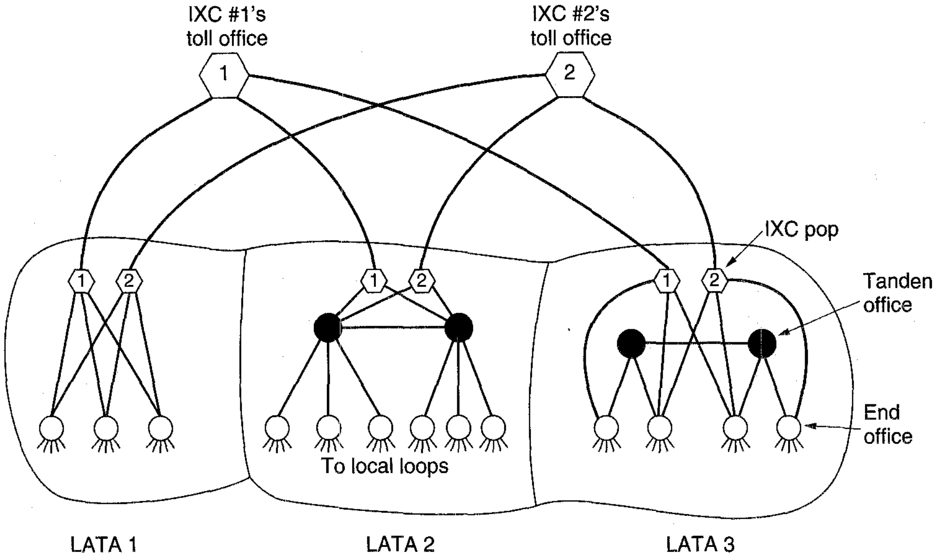


Fig. 2-16. The relationship of LATAs, LECs, and IXCs. All the circles are LEC switching offices. Each hexagon belongs to the IXC whose number is in it.

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.

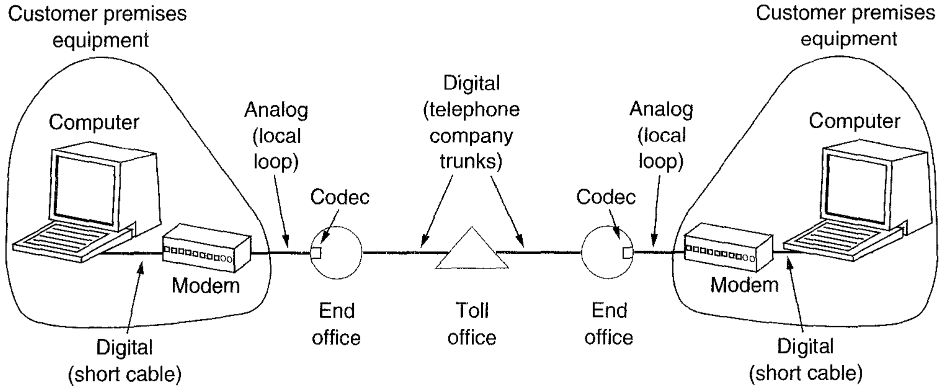


Fig. 2-17. The use of both analog and digital transmission for a computer to computer call. Conversion is done by the modems and codecs.

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.

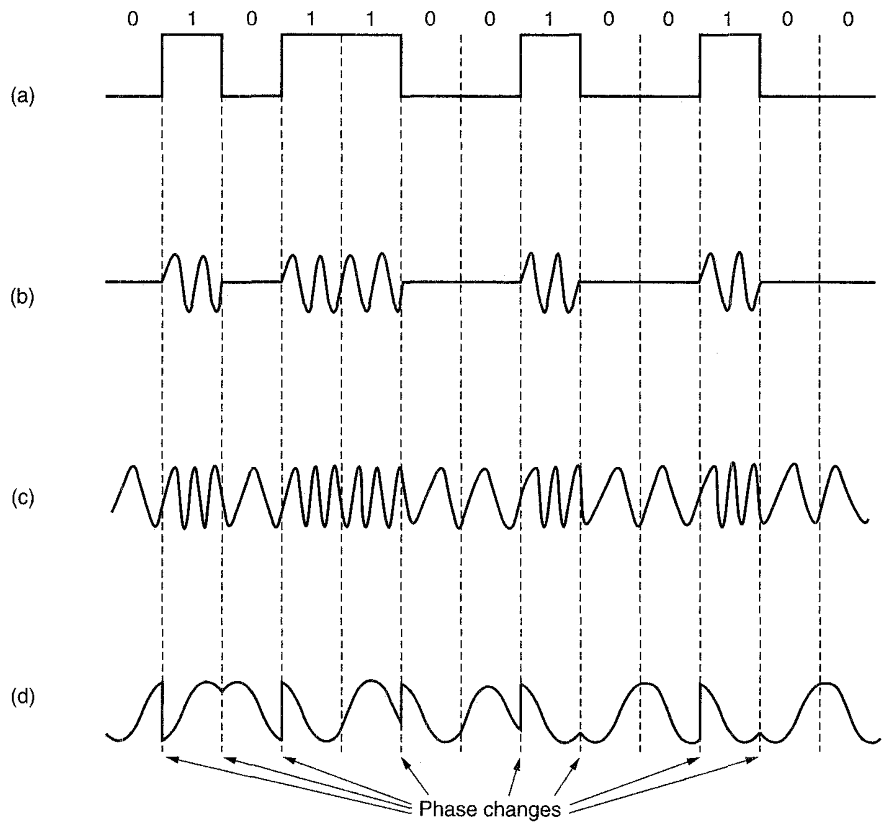


Fig. 2-18. (a) A binary signal. (b) Amplitude modulation. (c) Frequency modulation. (d) Phase modulation.

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**).

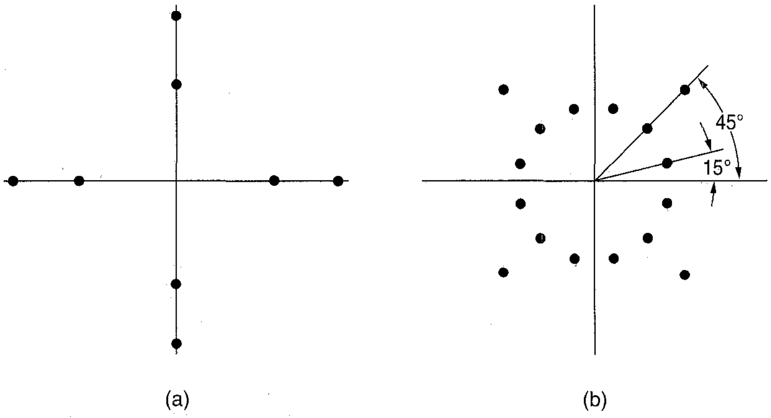


Fig. 2-19. (a) 3 bits/baud modulation. (b) 4 bits/baud modulation.

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

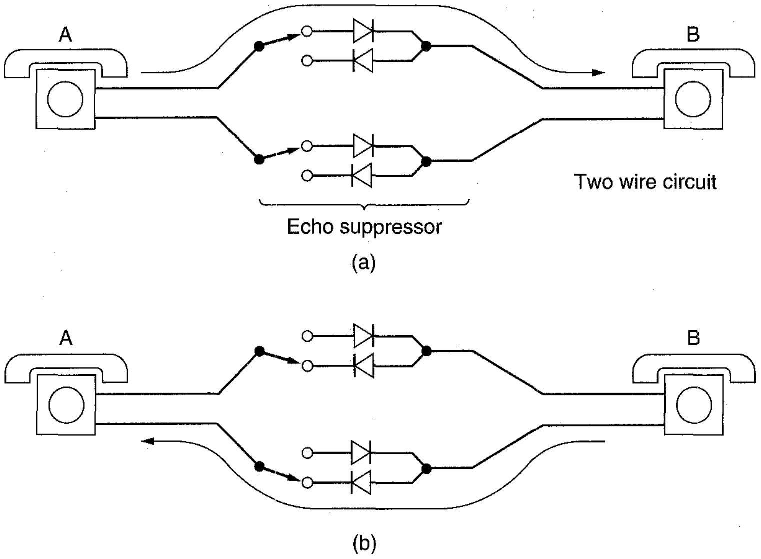


Fig. 2-20. (a) *A* talking to *B*. (b) *B* talking to *A*.

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.

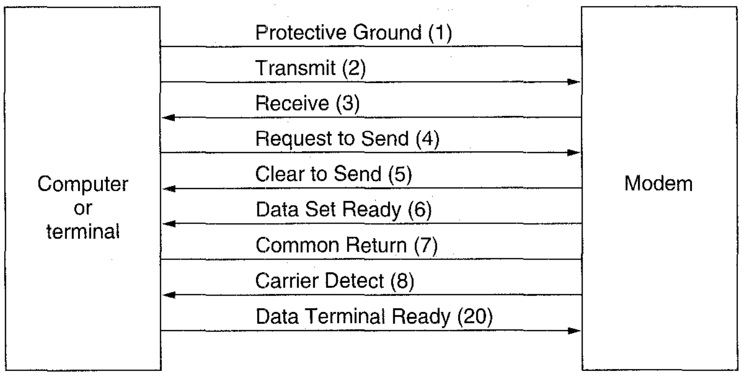


Fig. 2-21. Some of the principal RS-232-C circuits. The pin numbers are given in parentheses.

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).

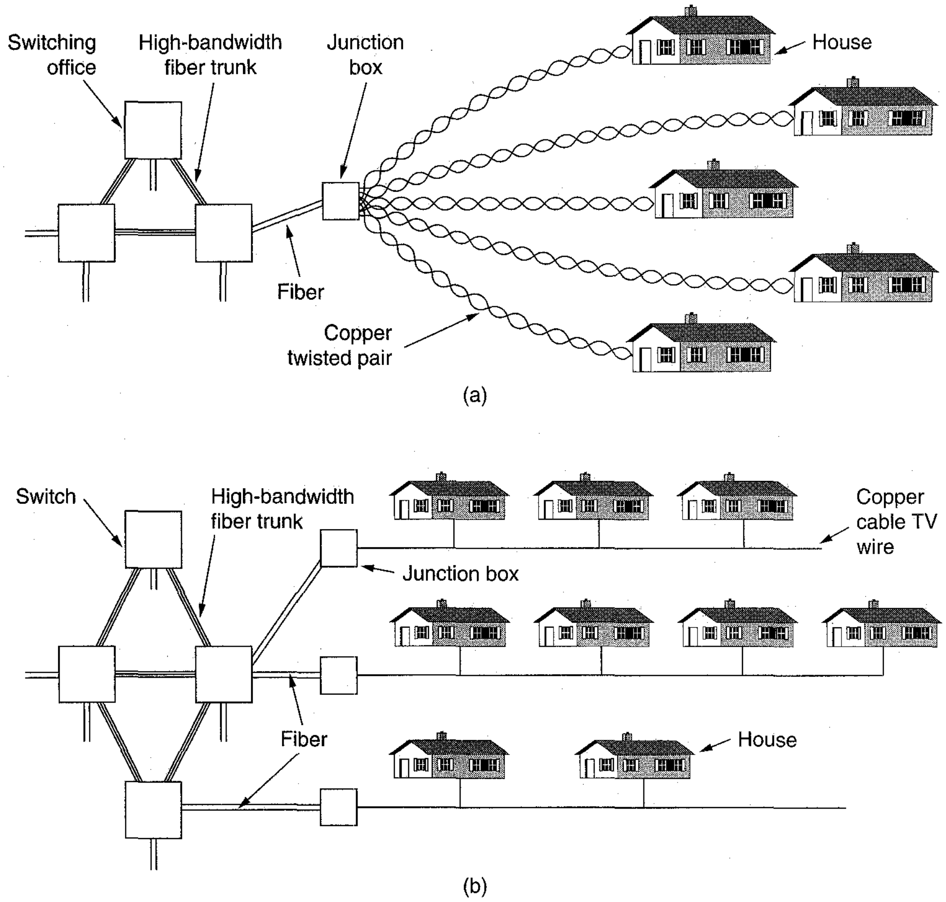


Fig. 2-23. Fiber to the curb. (a) Using the telephone network. (b) Using the cable TV network.

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 frequency, each by a different amount. Then they can be combined, because no two channels now occupy the same portion of the spectrum. Notice that even though there are gaps (guard bands) between the channels, there is some overlap between adjacent channels, because the filters do not have sharp edges. This overlap means that a strong spike at the edge of one channel will be felt in the adjacent one as nonthermal noise.

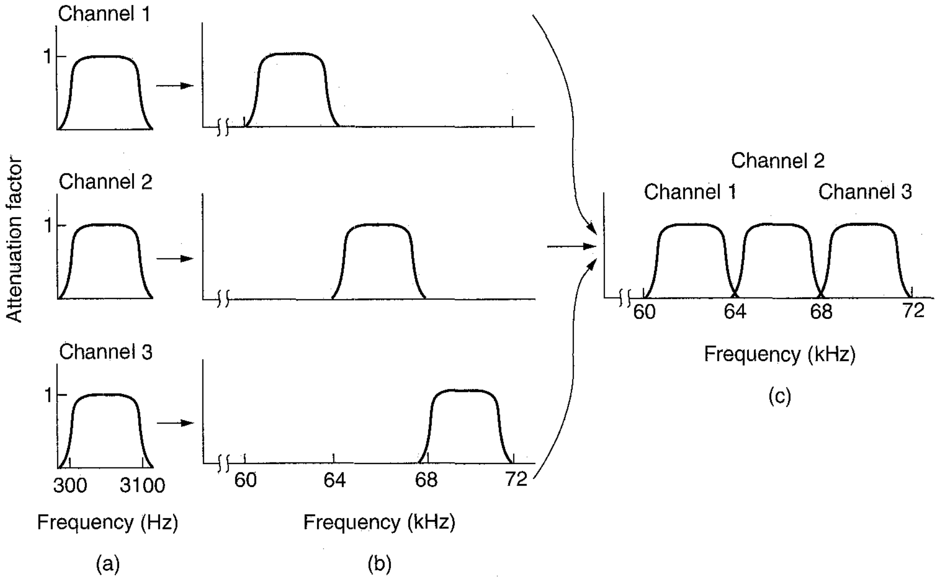


Fig. 2-24. Frequency division multiplexing. (a) The original bandwidths. (b) The bandwidths raised in frequency. (c) The multiplexed channel.

The FDM schemes used around the world are to some degree standardized. A widespread standard is 12 4000-Hz voice channels (3000 Hz for the user, plus two guard bands of 500 Hz each) multiplexed into the 60 to 108 kHz band. This unit is called a **group**. The 12- to 60-kHz band is sometimes used for another group. Many carriers offer a 48- to 56-kbps leased line service to customers, based on the group. Five groups (60 voice channels) can be multiplexed to form a **supergroup**. The next unit is the **mastergroup**, which is five supergroups (CCITT standard) or ten supergroups (Bell system). Other standards up to 230,000 voice channels also exist.

Wavelength Division Multiplexing

For fiber optic channels, a variation of frequency division multiplexing is used. It is called **WDM** (**Wavelength Division Multiplexing**). A simple way of achieving FDM on fibers is depicted in Fig. 2-25. Here two fibers come together at a prism (or more likely, a diffraction grating), each with its energy in a different band. The two beams are passed through the prism or grating, and combined onto a single shared fiber for transmission to a distant destination, where they are split again.

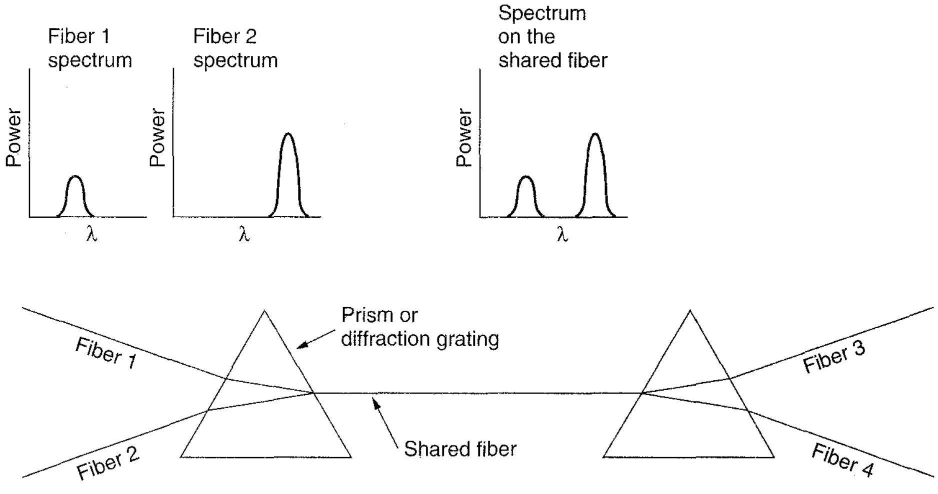


Fig. 2-25. Wavelength division multiplexing.

There is really nothing new here. As long as each channel has its own frequency range, and all the ranges are disjoint, they can be multiplexed together on the long-haul fiber. The only difference with electrical FDM is that an optical system using a diffraction grating is completely passive, and thus highly reliable.

It should be noted that the reason WDM is popular is that the energy on a single fiber is typically only a few gigahertz wide because it is currently impossible to convert between electrical and optical media any faster. Since the bandwidth of a single fiber band is about 25,000 GHz (see Fig. 2-6), there is great potential for multiplexing many channels together over long-haul routes. A necessary condition, however, is that the incoming channels use different frequencies.

A potential application of WDM is in the FTTC systems described earlier. Initially, a telephone company could run a single fiber from an end office to a neighborhood junction box where it met up with twisted pairs from the houses. Years later, when the cost of fiber is lower and the demand for it is higher, the twisted pairs can be replaced by fiber and all the local loops joined onto the fiber running to the end office using WDM.

In the example of Fig. 2-25, we have a fixed wavelength system. Bits from fiber 1 go to fiber 3, and bits from fiber 2 go to fiber 4. It is not possible to have bits go from fiber 1 to fiber 4. However, it is also possible to build WDM systems that are switched. In such a device, there are many input fibers and many output fibers, and the data from any input fiber can go to any output fiber. Typically, the coupler is a passive star, with the light from every input fiber illuminating the star. Although spreading the energy over *n* outputs dilutes it by a factor *n*, such systems are practical for hundreds of channels.

Of course, if the light from one of the incoming fibers is at 1.50206 microns and potentially might have to go to any output fiber, all the output fibers need tunable filters so the selected one can set itself to 1.50206 microns. Such optical tunable filters can be built from Fabry-Perot or Mach-Zehnder interferometers. Alternatively, the input fibers could be tunable and the output ones fixed. Having both be tunable is an unnecessary expense and is rarely worth it.

Time Division Multiplexing

Although FDM is still used over copper wires or microwave channels, it requires analog circuitry and is not amenable to being done by a computer. In contrast, TDM can be handled entirely by digital electronics, so it has become far more widespread in recent years. Unfortunately, it can only be used for digital data. Since the local loops produce analog signals, a conversion is needed from analog to digital in the end office, where all the individual local loops come together to be combined onto outgoing trunks. We will now look at how multiple analog voice signals are digitized and combined onto a single outgoing digital trunk. (Remember that computer data sent over a modem are also analog when they get to the end office.)

The analog signals are digitized in the end office by a device called a **codec** (coder-decoder), producing a 7- or 8-bit number (see Fig. 2-17). The codec makes 8000 samples per second (125 μsec/sample) because the Nyquist theorem says that this is sufficient to capture all the information from the 4-kHz telephone channel bandwidth. At a lower sampling rate, information would be lost; at a higher one, no extra information would be gained. This technique is called **PCM** (**Pulse Code Modulation**). PCM forms the heart of the modern telephone system. As a consequence, virtually all time intervals within the telephone system are multiples of 125 μsec.

When digital transmission began emerging as a feasible technology, CCITT was unable to reach agreement on an international standard for PCM. Consequently, there are now a variety of incompatible schemes in use in different countries around the world. International hookups between incompatible countries require (often expensive) "black boxes" to convert the originating country's system to that of the receiving country.

One method that is in widespread use in North America and Japan is the T1 carrier, depicted in Fig. 2-26. (Technically speaking, the format is called DS1 and the carrier is called T1, but we will not make that subtle distinction here.) The T1 carrier consists of 24 voice channels multiplexed together. Usually, the analog signals are sampled on a round-robin basis with the resulting analog stream being fed to the codec rather than having 24 separate codecs and then merging the digital output. Each of the 24 channels, in turn, gets to insert 8 bits into the output stream. Seven bits are data, and one is for control, yielding 7 × 8000 = 56,000 bps of data, and 1 × 8000 = 8000 bps of signaling information per channel.

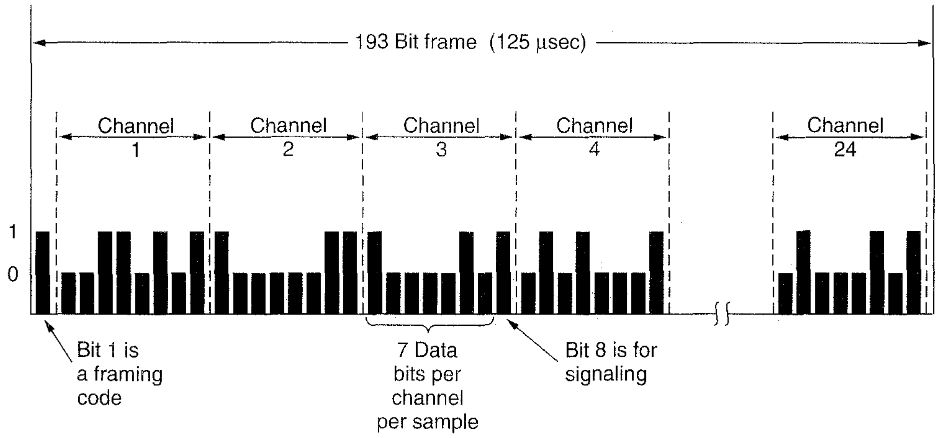


Fig. 2-26. The T1 carrier (1.544 Mbps).

A frame consists of 24 × 8 = 192 bits, plus one extra bit for framing, yielding 193 bits every 125 μsec. This gives a gross data rate of 1.544 Mbps. The 193rd bit is used for frame synchronization. It takes on the pattern 0101010101 . . . . Normally, the receiver keeps checking this bit to make sure that it has not lost synchronization. If it does get out of sync, the receiver can scan for this pattern to get resynchronized. Analog customers cannot generate the bit pattern at all, because it corresponds to a sine wave at 4000 Hz, which would be filtered out. Digital customers can, of course, generate this pattern, but the odds are against its being present when the frame slips. When a T1 system is being used entirely for data, only 23 of the channels are used for data. The 24th one is used for a special synchronization pattern, to allow faster recovery in the event that the frame slips.

When CCITT finally did reach agreement, they felt that 8000 bps of signaling information was far too much, so its 1.544-Mbps standard is based upon an 8- rather than a 7-bit data item; that is, the analog signal is quantized into 256 rather than 128 discrete levels. Two (incompatible) variations are provided. In **common-channel signaling**, the extra bit (which is attached onto the rear rather than the front of the 193 bit frame) takes on the values 10101010 . . . in the odd frames and contains signaling information for all the channels in the even frames.

In the other variation, **channel associated signaling**, each channel has its own private signaling subchannel. A private subchannel is arranged by allocating one of the eight user bits in every sixth frame for signaling purposes, so five out of six samples are 8 bits wide, and the other one is only 7 bits wide. CCITT also has a recommendation for a PCM carrier at 2.048 Mbps called **E1**. This carrier has 32 8-bit data samples packed into the basic 125-μsec frame. Thirty of the channels are used for information and two are used for signaling. Each group of four frames provides 64 signaling bits, half of which are used for channel associated signaling and half of which are used for frame synchronization or are reserved for each country to use as it wishes. Outside North America and Japan, the 2.048-Mbps carrier is in widespread use.

Once the voice signal has been digitized, it is tempting to try to use statistical techniques to reduce the number of bits needed per channel. These techniques are appropriate not only to encoding speech, but to the digitization of any analog signal. All of the compaction methods are based upon the principle that the signal changes relatively slowly compared to the sampling frequency, so that much of the information in the 7- or 8-bit digital level is redundant.

One method, called **differential pulse code modulation**, consists of outputting not the digitized amplitude, but the difference between the current value and the previous one. Since jumps of ±16 or more on a scale of 128 are unlikely, 5 bits should suffice instead of 7. If the signal does occasionally jump wildly, the encoding logic may require several sampling periods to "catch up." For speech, the error introduced can be ignored.

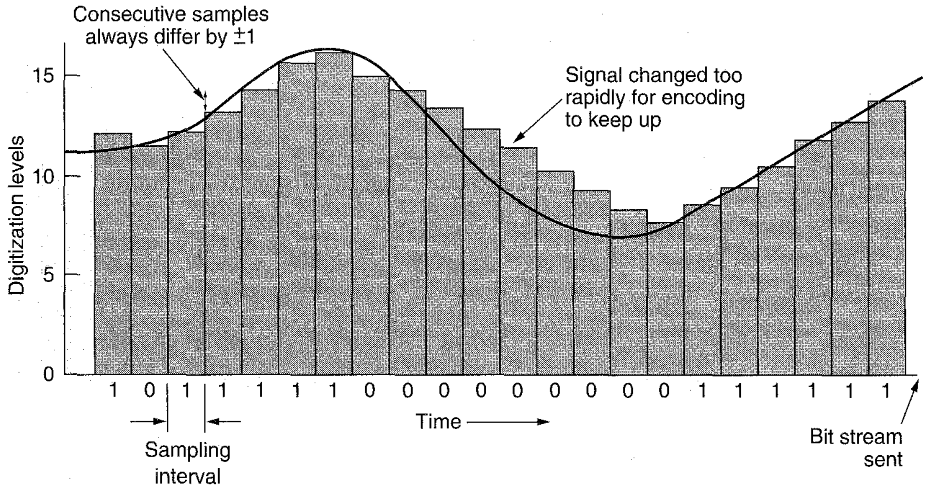


Fig. 2-27. Delta modulation.

A variation of this compaction method requires each sampled value to differ from its predecessor by either +1 or −1. A single bit is transmitted, telling whether the new sample is above or below the previous one. This technique, called **delta modulation**, is illustrated in Fig. 2-27. Like all compaction techniques that assume small level changes between consecutive samples, delta encoding can get into trouble if the signal changes too fast, as shown in the figure. When this happens, information is lost.

An improvement to differential PCM is to extrapolate the previous few values to predict the next value and then to encode the difference between the actual signal and the predicted one. The transmitter and receiver must use the same prediction algorithm, of course. Such schemes are called **predictive encoding**. They are useful because they reduce the size of the numbers to be encoded, hence the number of bits to be sent.

Although PCM is widely used on interoffice trunks, the computer user gets relatively little benefit from it if all data must be sent to the end office in the form of a modulated analog sine wave at 28.8 kbps. It would be nice if the carrier would attach the local loop directly to the PCM trunk system, so that the computer could output digital data directly onto the local loop at 1.544 or 2.048 Mbps. Unfortunately, the local loops cannot run at these speeds for very far.

Time division multiplexing allows multiple T1 carriers to be multiplexed into higher-order carriers. Figure 2-28 shows how this can be done. At the left we see four T1 channels being multiplexed onto one T2 channel. The multiplexing at T2 and above is done bit for bit, rather than byte for byte with the 24 voice channels that make up a T1 frame. Four T1 streams at 1.544 Mbps should generate 6.176 Mbps, but T2 is actually 6.312 Mbps. The extra bits are used for framing and recovery, in case the carrier slips.

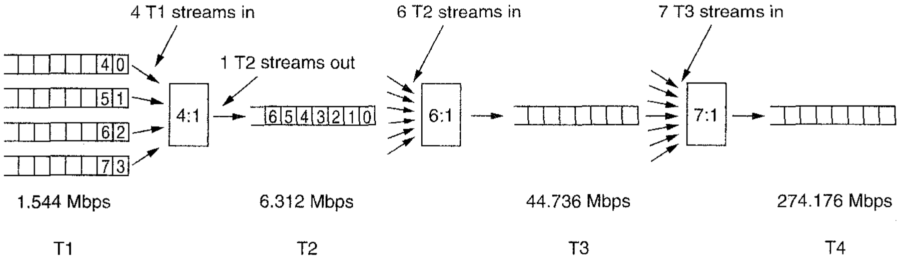


Fig. 2-28. Multiplexing T1 streams onto higher carriers.

At the next level, six T2 streams are combined bitwise to form a T3 stream. Then seven T3 streams are joined to form a T4 stream. At each step a small amount of overhead is added for framing and recovery.

Just as there is little agreement on the basic carrier between the United States and the rest of the world, there is equally little agreement on how it is to be multiplexed into higher bandwidth carriers. The U.S. scheme of stepping up by 4, 6, and 7 did not strike everyone else as the way to go, so the CCITT standard calls for multiplexing four streams onto one stream at each level. Also, the framing and recovery data are different. The CCITT hierarchy for 32, 128, 512, 2048, and 8192 channels runs at speeds of 2.048, 8.848, 34.304, 139.264, and 565.148 Mbps.

SONET/SDH

In the early days of fiber optics, every telephone company had its own proprietary optical TDM system. After AT&T was broken up in 1984, local telephone companies had to connect to multiple long-distance carriers, all with different optical TDM systems, so the need for standardization became obvious. In 1985, Bellcore, the RBOCs research arm, began working on a standard, called **SONET (Synchronous Optical NETwork)**. Later, CCITT joined the effort, which resulted in a SONET standard and a set of parallel CCITT recommendations (G.707, G.708, and G.709) in 1989. The CCITT recommendations are called **SDH (Synchronous Digital Hierarchy)** but differ from SONET only in minor ways. Virtually all the long-distance telephone traffic in the United States, and much of it elsewhere now uses trunks running SONET in the physical layer. As SONET chips become cheaper, SONET interface boards for computers may become more widespread, so it may become easier for companies to plug their computers directly into the heart of the telephone network over specially conditioned leased lines. Below we will discuss the goals and design of SONET briefly. For additional information see (Bellamy, 1991; and Omidyar and Aldridge, 1993).

The SONET design had four major goals. First and foremost, SONET had to make it possible for different carriers to interwork. Achieving this goal required defining a common signaling standard with respect to wavelength, timing, framing structure, and other issues.

Second, some means was needed to unify the U.S., European, and Japanese digital systems, all of which were based on 64-kbps PCM channels, but all of which combined them in different (and incompatible) ways.

Third, SONET had to provide a way to multiplex multiple digital channels together. At the time SONET was devised, the highest speed digital carrier actually used widely in the United States was T3, at 44.736 Mbps. T4 was defined, but not used much, and nothing was even defined above T4 speed. Part of SONET's mission was to continue the hierarchy to gigabits/sec and beyond. A standard way to multiplex slower channels into one SONET channel was also needed.

Fourth, SONET had to provide support for operations, administration, and maintenance (OAM). Previous systems did not do this very well.

An early decision was to make SONET a traditional TDM system, with the entire bandwidth of the fiber devoted to one channel containing time slots for the various subchannels. As such, SONET is a synchronous system. It is controlled by a master clock with an accuracy of about 1 part in 10^9. Bits on a SONET line are sent out at extremely precise intervals, controlled by the master clock.

When cell switching was later proposed to be the basis of broadband ISDN, the fact that it permitted irregular cell arrivals got it labeled as *asynchronous* transfer mode (i.e., ATM) to contrast it to the synchronous operation of SONET.

A SONET system consists of switches, multiplexers, and repeaters, all connected by fiber. A path from a source to destination with one intermediate multiplexer and one intermediate repeater is shown in Fig. 2-29. In SONET terminology, a fiber going directly from any device to any other device, with nothing in between, is called a **section**. A run between two multiplexers (possibly with one or more repeaters in the middle) is called a **line**. Finally, the connection between the source and destination (possibly with one or more multiplexers and repeaters) is called a **path**. The SONET topology can be a mesh, but is often a dual ring.

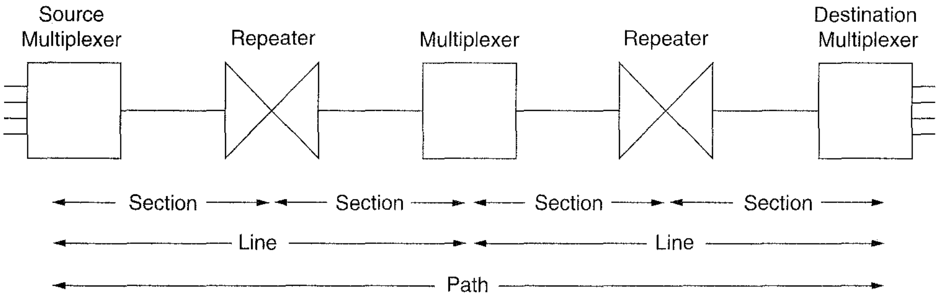


Fig. 2-29. A SONET path.

The basic SONET frame is a block of 810 bytes put out every 125 μsec. Since SONET is synchronous, frames are emitted whether or not there are any useful data to send. Having 8000 frames/sec exactly matches the sampling rate of the PCM channels used in all digital telephony systems.

The 810-byte SONET frames are best described as a rectangle of bytes, 90 columns wide by 9 rows high. Thus $8 \times 810 = 6480$ bits are transmitted 8000 times per second, for a gross data rate of 51.84 Mbps. This is the basic SONET channel and is called **STS-1** (**Synchronous Transport Signal-1**). All SONET trunks are a multiple of STS-1.

The first three columns of each frame are reserved for system management information, as illustrated in Fig. 2-30. The first three rows contain the section overhead; the next six contain the line overhead. The section overhead is generated and checked at the start and end of each section, whereas the line overhead is generated and checked at the start and end of each line.

The remaining 87 columns hold $87 \times 9 \times 8 \times 8000 = 50.112$ Mbps of user data. However, the user data, called the **SPE** (**Synchronous Payload Envelope**) do not always begin in row 1, column 4. The SPE can begin anywhere within the frame. A pointer to the first byte is contained in the first row of the line overhead. The first column of the SPE is the path overhead (i.e., header for the end-to-end path sublayer protocol).

The ability to allow the SPE to begin anywhere within the SONET frame, and even to span two frames, as shown in Fig. 2-30, gives added flexibility to the

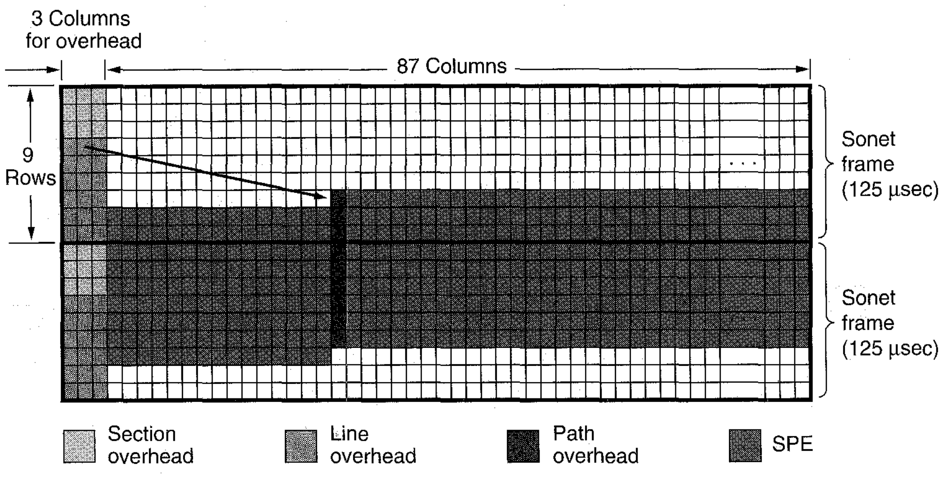


Fig. 2-30. Two back-to-back SONET frames.

system. For example, if a payload arrives at the source while a dummy SONET frame is being constructed, it can be inserted into the current frame, instead of being held until the start of the next one. This feature is also useful when the payload does not fit exactly in one frame, as in the case of a sequence of 53-byte ATM cells. The first row of the line overhead can then point to the start of the first full cell, to provide synchronization.

The section, line, and path overheads contain a profusion of bytes used for operations, administration, and maintenance. Since each byte occurs 8000 times per second, it represents a PCM channel. Three of these are, in fact, used to provide voice channels for section, line, and path maintenance personnel. Other bytes are used for framing, parity, error monitoring, IDs, clocking, synchronization, and other functions. Bellamy (1991) describes all the fields in detail.

The multiplexing of multiple data streams, called **tributaries**, plays an important role in SONET. Multiplexing is illustrated in Fig. 2-31. On the left, we start with various low-speed input streams, which are converted to the basic STS-1 SONET rate, in most cases by adding filler to round up to 51.84 Mbps. Next, three STS-1 tributaries are multiplexed onto one 155.52-Mbps STS-3 output stream. This stream, in turn, is multiplexed with three others onto a final output stream having 12 times the capacity of the STS-1 stream. At this point the signal is scrambled, to prevent long runs of 0s or 1s from interfering with the clocking, and converted from an electrical to an optical signal.

Multiplexing is done byte for byte. For example, when three STS-1 tributaries at 51.84 Mbps are merged into one STS-3 stream at 155.52 Mbps, the multiplexer first outputs 1 byte from tributary 1, then 1 from tributary 2, and finally 1 from tributary 3, before going back to 1. The STS-3 figure analogous to Fig. 2-30

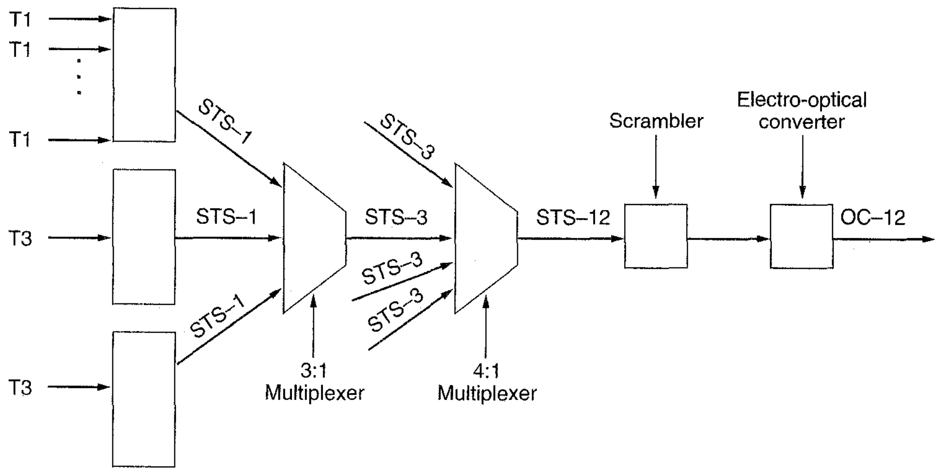


Fig. 2-31. Multiplexing in SONET.

shows (from left to right) columns from tributaries 1, 2, and 3, in that order, then another triple, and so on, out to column 270. One of these 270 × 9 byte frames is sent every 125 μsec, giving the 155.52-Mbps data rate.

The SONET multiplexing hierarchy is shown in Fig. 2-32. Rates from STS-1 to STS-48 have been defined. The optical carrier corresponding to STS-*n* is called OC-*n* but is bit-for-bit the same except for the scrambling shown in Fig. 2-31. The SDH names are different, and they start at OC-3 because CCITT-based systems do not have a rate near 51.84 Mbps. The OC-9 carrier is present because it closely matches the speed of a major high-speed trunk used in Japan. OC-18 and OC-36 will be used in Japan in the future. The gross data rate includes all the overhead. The SPE data rate excludes the line and section overhead. The user data rate excludes all overhead and only counts the 86 columns available for the payload.

As an aside, when a carrier, such as OC-3, is not multiplexed, but carries the data from only a single source, the letter *c* (for concatenated) is appended to the designation, so OC-3 indicates a 155.52-Mbps carrier consisting of three separate OC-1 carriers, but OC-3c indicates a data stream from a single source at 155.52 Mbps. The three OC-1 streams within an OC-3c stream are interleaved by column, first column 1 from stream 1, then column 1 from stream 2, then column 1 from stream 3, followed by column 2 from stream 1, and so on, leading to a frame 270 columns wide and 9 rows deep.

The amount of actual user data in an OC-3c stream is slightly higher than in an OC-3 stream (149.760 Mbps versus 148.608 Mbps) because the path overhead column is included inside the SPE only once, instead of the three times it would be with three independent OC-1 streams. In other words, 260 of the 270 columns

SONET		SDH	Data rate (Mbps)		
Electrical	**Optical**	**Optical**	**Gross**	**SPE**	**User**
STS-1	OC-1		51.84	50.112	49.536
STS-3	OC-3	STM-1	155.52	150.336	148.608
STS-9	OC-9	STM-3	466.56	451.008	445.824
STS-12	OC-12	STM-4	622.08	601.344	594.432
STS-18	OC-18	STM-6	933.12	902.016	891.648
STS-24	OC-24	STM-8	1244.16	1202.688	1188.864
STS-36	OC-36	STM-12	1866.24	1804.032	1783.296
STS-48	OC-48	STM-16	2488.32	2405.376	2377.728

Fig. 2-32. SONET and SDH multiplex rates.

are available for user data in OC-3c, whereas only 258 columns are available for user data in OC-3. Higher-order concatenated frames (e.g., OC-12c) also exist.

By now it should be clear why ATM runs at 155 Mbps: the intention is to carry ATM cells over SONET OC-3c trunks. It should also be clear that the widely quoted 155-Mbps figure is the gross rate, including the SONET overhead. Furthermore, somewhere along the way somebody incorrectly rounded 155.52 Mbps to 155 Mbps instead of 156 Mbps, and now everyone else does it wrong, too.

The SONET physical layer is divided up into four sublayers, as shown in Fig. 2-33. The lowest sublayer is the **photonic sublayer**. It is concerned with specifying the physical properties of the light and fiber to be used.

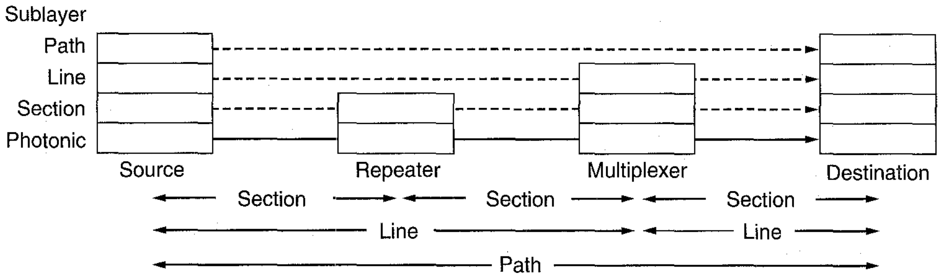


Fig. 2-33. The SONET architecture.

The three remaining sublayers correspond to the sections, lines, and paths. The section sublayer handles a single point-to-point fiber run, generating a standard frame at one end and processing it at the other. Sections can start and end at 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.

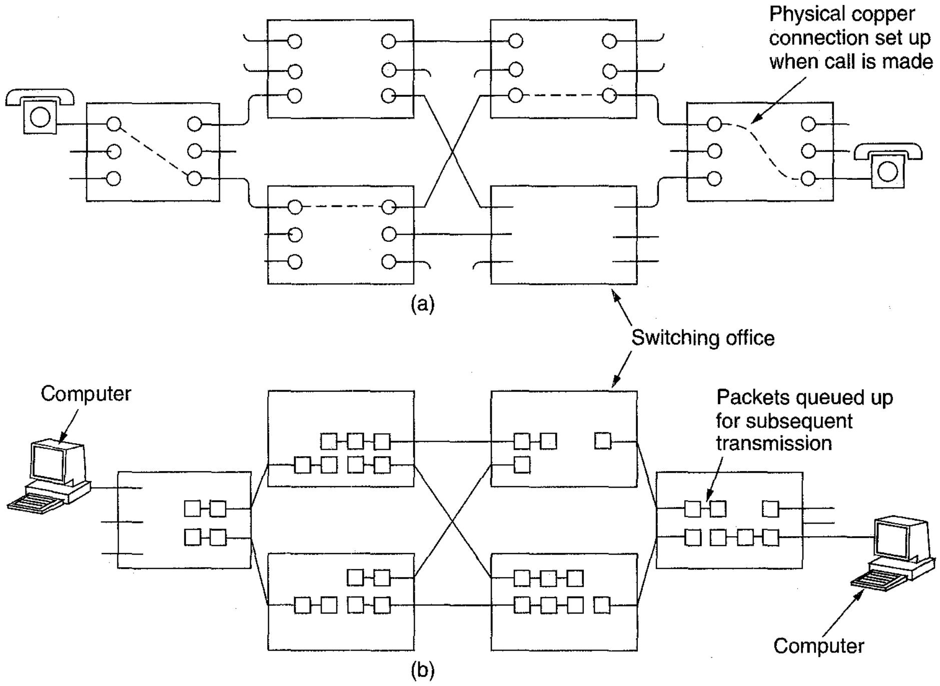


Fig. 2-34. (a) Circuit switching. (b) Packet switching.

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.

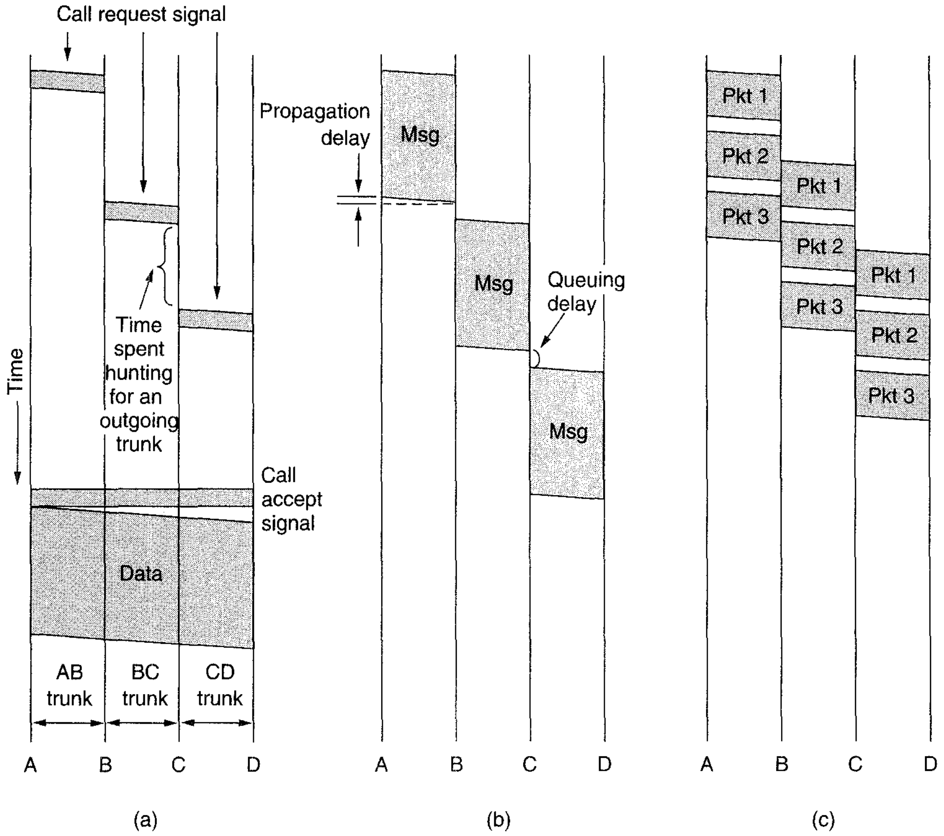


Fig. 2-35. Timing of events in (a) circuit switching, (b) message switching, (c) packet switching.

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

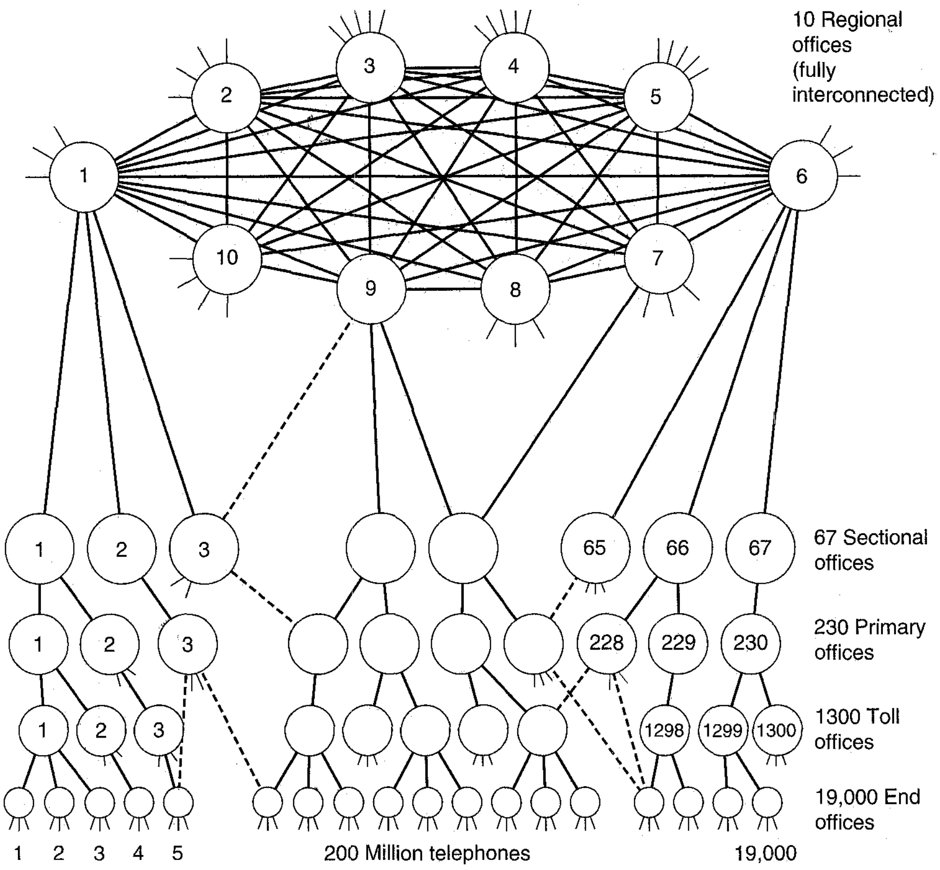


Fig. 2-37. The AT&T telephone hierarchy. The dashed lines are direct trunks.

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

intersections, called **crosspoints**, where an input and an output line may be connected by a semiconductor switch, as shown in Fig. 2-38(a). In Fig. 2-38(b) we see an example in which line 0 is connected to line 4, line 1 is connected to line 7, and line 2 is connected to line 6. Lines 3 and 5 are not connected. All the bits that arrive at the switch from line 4, for example, are immediately sent out of the switch on line 0. Thus the crossbar switch implements circuit switching by making a direct electrical connection, just like the jumper cables in the first-generation switches, only automatically and within microseconds.

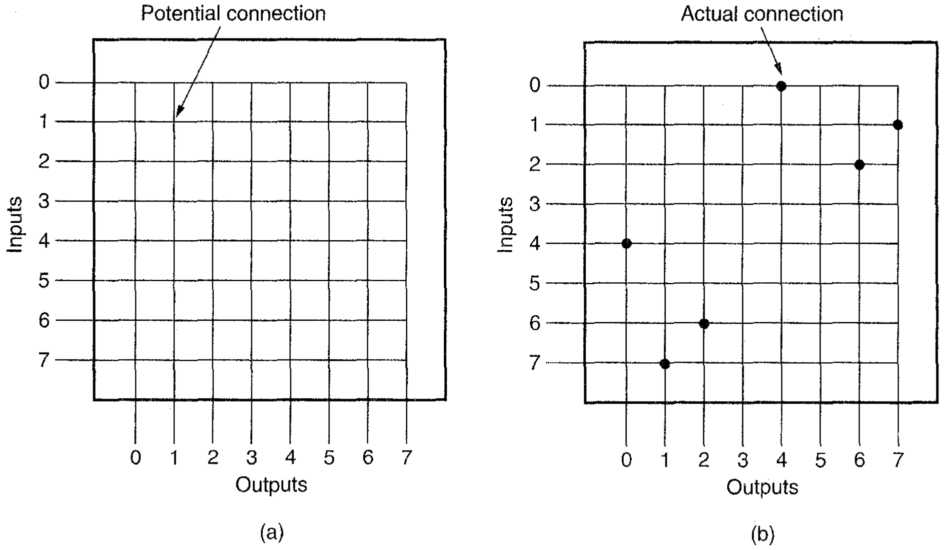


Fig. 2-38. (a) A crossbar switch with no connections. (b) A crossbar switch with three connections set up: 0 with 4, 1 with 7, and 2 with 6.

The problem with a crossbar switch is that the number of crossbars grows as the square of the number of lines into the switch. If we assume that all lines are full duplex and that there are no self-connections, only the crosspoints above the diagonal are needed. Still, $n(n-1)/2$ crosspoints are needed. For $n = 1000$, we need 499,500 crosspoints. While building a VLSI chip with this number of transistor switches is possible, having 1000 pins on the chip is not. Thus a single crossbar switch is only useful for relatively small end offices.

Space Division Switches

By splitting the crossbar switch into small chunks and interconnecting them, it is possible to build multistage switches with many fewer crosspoints. These are called **space division switches**. Two configurations are illustrated in Fig. 2-39.

To keep our example simple, we will consider only three-stage switches, but

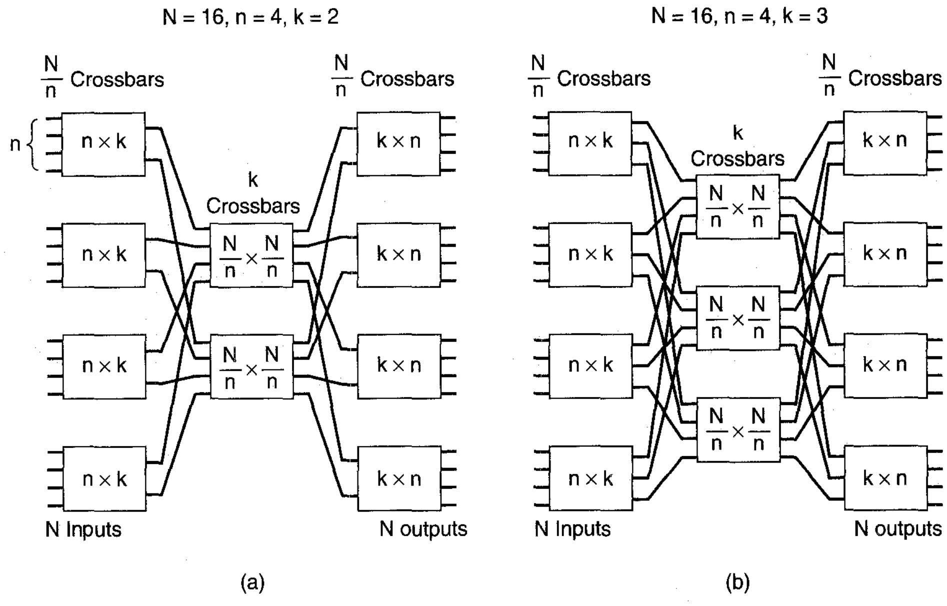


Fig. 2-39. Two space division switches with different parameters.

switches with more stages are also possible. In these examples, we have a total of N inputs and N outputs. Instead of building a single $N \times N$ crossbar, we build the switch out of smaller rectangular crossbars. In the first stage, each crossbar has n inputs, so we need N/n of them to handle all N incoming lines.

The second stage has k crossbars, each with N/n inputs and N/n outputs. The third stage is a repeat of the first stage, but reversed left to right. Each intermediate crossbar is connected to each input crossbar and each output crossbar. Consequently, it is possible to connect every input to every output using either the first intermediate crossbar in Fig. 2-39(a) or using the second one. In fact, there are two disjoint paths from each input to each output, depending which intermediate crossbar is chosen. In Fig. 2-39(b) there are three paths for each input/output pair. With k intermediate stages (k is a design parameter), there are k disjoint paths.

Let us now compute the number of crosspoints needed for a three-stage switch. In the first stage, there are N/n crossbars, each with nk crosspoints, for a total of Nk. In the second stage, there are k crossbars, each with $(N/n)^2$ crosspoints. The third stage is the same as the first. Adding up the three stages, we get

$$\text{Number of crosspoints} = 2kN + k(N/n)^2$$

For $N = 1000$, $n = 50$ and $k = 10$, we need only 24,000 crosspoints instead of the 499,500 required by a 1000×1000 single-stage crossbar.

Unfortunately, as usual, there is no free lunch. The switch can block. Consider Fig. 2-39(a) again. Stage 2 has eight inputs, so a maximum of eight calls can be connected at once. When call nine comes by, it will have to get a busy signal, even though the destination is available. The switch of Fig. 2-39(b) is better, handling a maximum of 12 calls instead of 8, but it uses more crosspoints. Sometimes when making a phone call you may have gotten a busy signal before you finished dialing. This was probably caused by blocking part way through the network.

It should be obvious that the larger *k* is, the more expensive the switch and the lower the blocking probability. In 1953, Clos showed that when $k = 2n - 1$, the switch will never block (Clos, 1953). Other researchers have analyzed calling patterns in great detail to construct switches that theoretically can block but do so only rarely in practice.

Time Division Switches

A completely different kind of switch is the **time division switch**, shown in Fig. 2-40. With time division switching, the *n* input lines are scanned in sequence to build up an input frame with *n* slots. Each slot has *k* bits. For T1 switches, the slots are 8 bits, with 8000 frames processed per second.

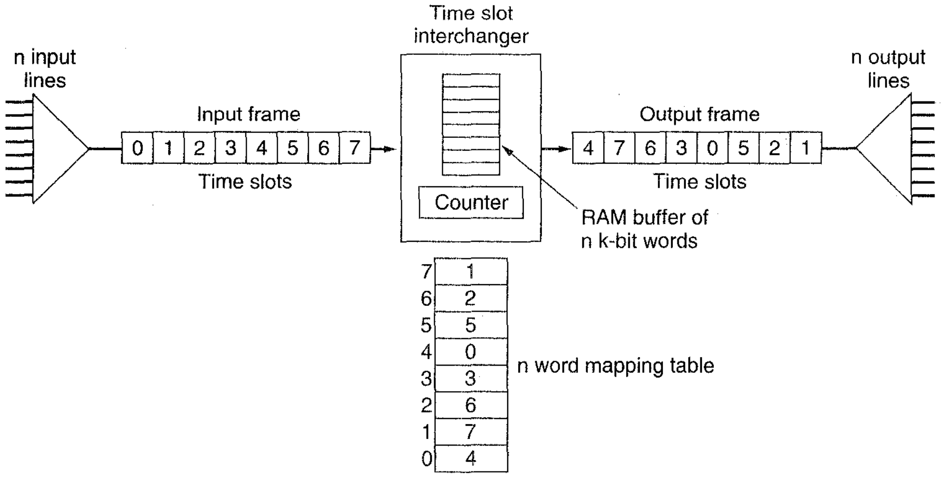


Fig. 2-40. A time division switch.

The heart of the time division switch is the **time slot interchanger**, which accepts input frames and produces output frames in which the time slots have been reordered. In Fig. 2-40, input slot 4 is output first, then slot 7, and so on. Finally, the output frame is demultiplexed, with output slot 0 (input slot 4) going 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.

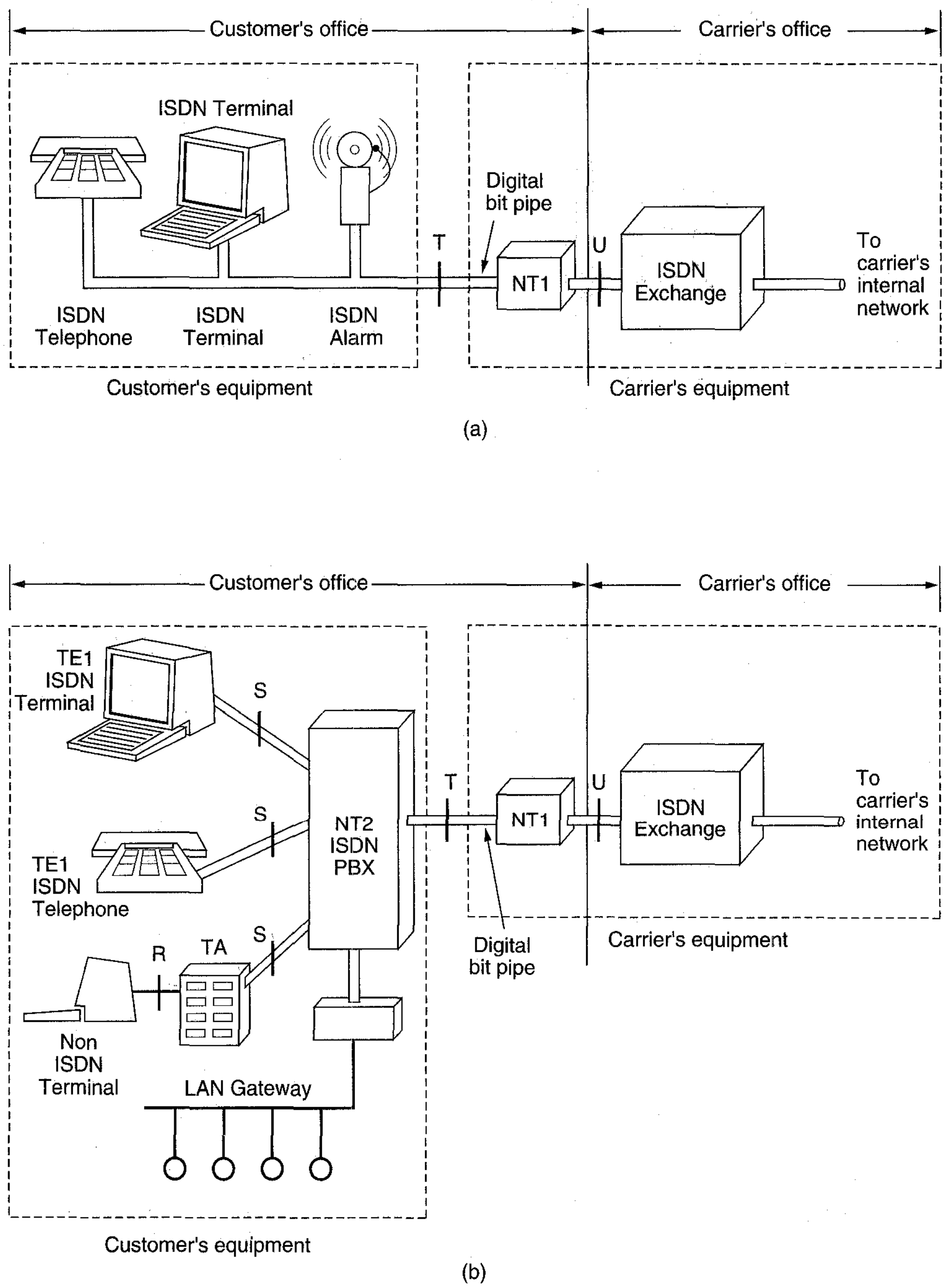


Fig. 2-41. (a) Example ISDN system for home use. (b) Example ISDN system with a PBX for use in large businesses.

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).

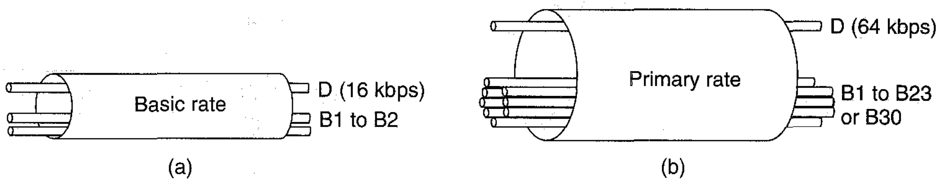


Fig. 2-42. (a) Basic rate digital pipe. (b) Primary rate digital pipe.

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.

2.6.1. Virtual Circuits versus Circuit Switching

The basic broadband ISDN service is a compromise between pure circuit switching and pure packet switching. The actual service offered is connection oriented, but it is implemented internally with packet switching, not circuit switching. Two kinds of connections are offered: permanent virtual circuits and switched virtual circuits. **Permanent virtual circuits** are requested by the customer manually (e.g., by sending a fax to the carrier) and typically remain in place for months or years. **Switched virtual circuits** are like telephone calls: they are set up dynamically as needed and potentially torn down immediately afterward.

In a circuit-switching network, making a connection actually means a physical path is established from the source to the destination through the network, certainly when space division switches are used. (With time division switches, the concept of "a physical path" is already getting a little fuzzy around the edges.) In a virtual circuit network, like ATM, when a circuit is established, what really happens is that the route is chosen from source to destination, and all the switches (i.e., routers) along the way make table entries so they can route any packets on that virtual circuit. They also have the opportunity to reserve resources for the new circuit. Figure 2-43 shows a single virtual circuit from host H_1 to host H_5 via switches (routers) *A*, *E*, *C*, and *D*.

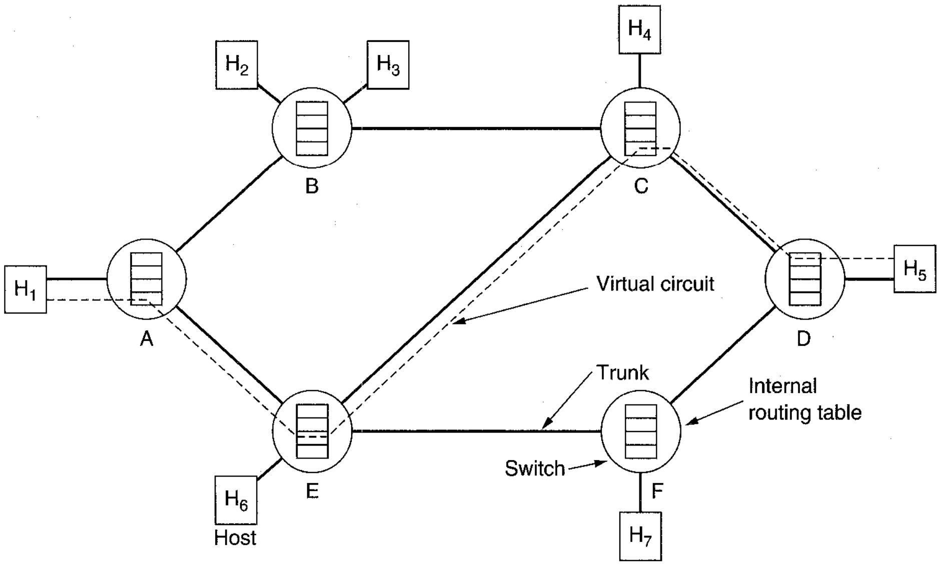


Fig. 2-43. The dotted line shows a virtual circuit. It is simply defined by table entries inside the switches.

When a packet comes along, the switch inspects the packet's header to find out which virtual circuit it belongs to. Then it looks up that virtual circuit in its tables to determine which communication line to send on. We will examine this process in more detail in Chap. 5.

The meaning of the permanent virtual circuit between H_1 and H_5 in Fig. 2-43 should now be clear. It is an agreement between the customer and the carrier that the switches will always hold table entries for a particular destination, even if there has been no traffic for months. Clearly, such an agreement costs resources (certainly table space inside the switches and possibly reserved bandwidth and buffers as well) so there is always a monthly charge per permanent virtual circuit. The advantage over a switched virtual circuit is that there is no setup time. Packets can move instantly. For some applications, such as credit card verification, saving a few seconds on each transaction may easily be worth the cost.

In contrast, a leased line from H_1 to H_5 in a circuit-switched network with the topology of Fig. 2-43 and space division switches would actually hold the crosspoints closed for months and would actually reserve bandwidth on the trunks permanently, either as FDM bands or as time slots (a leased "line" can be multihop if no direct line is available). Such an arrangement is obviously far more wasteful of resources when it is idle than a virtual circuit.

2.6.2. Transmission in ATM Networks

As we have pointed out before, ATM stands for *Asynchronous* Transfer Mode. This mode can be contrasted with the synchronous T1 carrier illustrated in Fig. 2-44(a). One T1 frame is generated precisely every 125 μsec. This rate is governed by a master clock. Slot *k* of each frame contains 1 byte of data from the same source. T1 is synchronous.

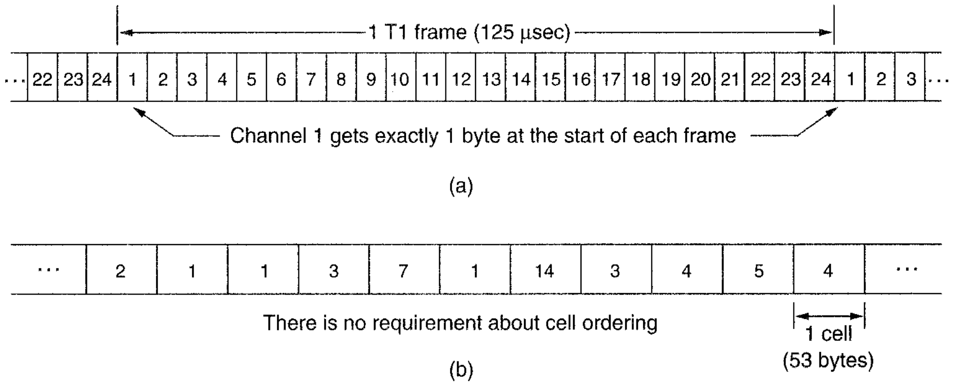


Fig. 2-44. (a) Synchronous transmission mode. (b) Asynchronous transmission mode.

ATM, in contrast, has no requirement that cells rigidly alternate among the various sources. Fig. 2-44(b) shows cells on a line from various sources, with no particular pattern. Cells arrive randomly from different sources.

Furthermore, it is not even required that the stream of cells coming out of a computer be continuous. Gaps between the data cells are possible. Such gaps are filled by special idle cells.

ATM does not standardize the format for transmitting cells. Rather, it specifies that just sending individual cells is allowed but also specifies that cells may be encased in a carrier such as T1, T3, SONET, or FDDI (a fiber optic LAN). For these examples, standards exist telling how cells are packed into the frames these systems provide.

In the original ATM standard, the primary rate was 155.52 Mbps, with an additional rate at four times that speed (622.08 Mbps). These rates were chosen to be compatible with SONET, the framing standard used on fiber optic links throughout the telephone system. ATM over T3 (44.736 Mbps) and FDDI (100 Mbps) is also foreseen.

The transmission medium for ATM is normally fiber optics, but for runs under 100 meters, coax or category 5 twisted pair are also acceptable. Fiber runs can be many kilometers. Each link goes between a computer and an ATM switch, or between two ATM switches. In other words, all ATM links are point-to-point (unlike LANs, which have many senders and receivers on the same cable). Multicasting is achieved by having a cell enter a switch on one line and exit it on multiple lines. Each point-to-point link is unidirectional. For full-duplex operation, two parallel links are needed, one for traffic each way.

The ATM **Physical Medium Dependent** sublayer is concerned with getting the bits on and off the wire. Different hardware is needed for different cables and fibers, depending on the speed and line encoding. The purpose of the transmission convergence sublayer is to provide a uniform interface to the ATM layer in both directions. Outbound, the ATM layer provides a sequence of cells, and the PMD sublayer encodes them as necessary and pushes them out the door as a bit stream.

Inbound, the PMD sublayer takes the incoming bits from the network and delivers a bit stream to the TC sublayer. The cell boundaries are not marked in any way. It is up to the TC sublayer to somehow figure out how to tell where one cell ends and the next one begins. This job is not only difficult, it is theoretically impossible. Thus the TC sublayer clearly has its work cut out for it. Because the TC sublayer is doing cell framing, it is a data link function, so we will discuss it in Chap. 3. For additional information about the ATM physical layer, see (Rao and Hatamian, 1995).

2.6.3. ATM Switches

Many ATM cell switch designs have been described in the literature. Some of these have been implemented and tested. In this section we will give a brief introduction to the principles of ATM cell switch design and illustrate these with a few examples. For more information, see (De Prycker 1993; Garcia-Haro and

Jajszczyk, 1994; Handel et al., 1994; and Partridge, 1994). For an ATM switch optimized for running IP over ATM, see (Parulkar et al., 1995).

The general model for an ATM cell switch is shown in Fig. 2-45. It has some number of input lines and some number of output lines, almost always the same number (because the lines are bidirectional). ATM switches are generally synchronous in the sense of during a cycle, one cell is taken from each input line (if one is present), passed into the internal **switching fabric**, and eventually transmitted on the appropriate output line.

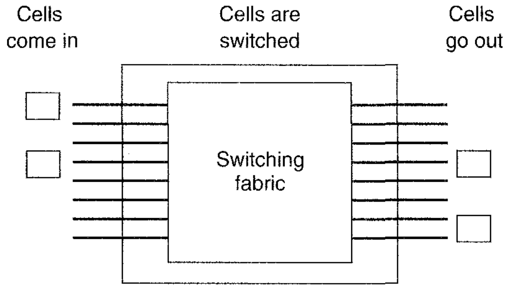


Fig. 2-45. A generic ATM switch.

Switches may be pipelined, that is, it may take several cycles before an incoming cell appears on its output line. Cells actually arrive on the input lines asynchronously, so there is a master clock that marks the beginning of a cycle. Any cell fully arrived when the clock ticks is eligible for switching during that cycle. A cell not fully arrived has to wait until the next cycle.

Cells arrive at ATM speed, normally about 150 Mbps. This works out to slightly over 360,000 cells/sec, which means that the cycle time of the switch has to be about 2.7 μsec. A commercial switch might have anywhere from 16 to 1024 input lines, which means that it must be prepared to accept and start switching a batch of 16 to 1024 cells every 2.7 μsec. At 622 Mbps, a new batch of cells is injected into the switching fabric about every 700 nsec. The fact that the cells are fixed length and short (53 bytes) makes it possible to build such switches. With longer variable-length packets, high-speed switching would be more complex, which is why ATM uses short fixed-length cells.

All ATM switches have two common goals:

1. Switch all cells with as low a discard rate as possible.
2. Never reorder the cells on a virtual circuit.

Goal 1 says that it is permitted to drop cells in emergencies, but that the loss rate should be as small as possible. A loss rate of 1 cell in 10^{12} is probably acceptable. On a large switch, this loss rate is about 1 or 2 cells per hour. Goal 2 says that cells arriving on a virtual circuit in a certain order must also depart in that order, with no exceptions, ever. This constraint makes switch design considerably more difficult, but it is required by the ATM standard.

A problem that occurs in all ATM switches is what to do if the cells arriving at two or more input lines want to go to the same output port in the same cycle. Solving this problem is one of the key issues in the design of all ATM switches. One nonsolution is to pick one cell to deliver and discard the rest. Since this algorithm violates goal 1, we cannot use it.

Our next attempt is to provide a queue for each input line. If two or more cells conflict, one of them is chosen for delivery, and the rest are held for the next cycle. The choice can be made at random, or cyclically, but should not exhibit systematic bias in favor of, for example, the lowest-numbered line to avoid giving lines with low numbers better service than lines with high numbers. Figure 2-46(a) depicts the situation at the start of cycle 1, in which cells have arrived on all four input lines, destined for output lines 2, 0, 2, and 1, respectively. Because there is a conflict for line 2, only one of the cells can be chosen. Suppose that it is the one on input line 0. At the start of cycle 1, shown in Fig. 2-46(b), three cells have been output, but the cell on line 2 has been held, and two more cells have arrived. Only at the start of cycle 4, shown in Fig. 2-46(d), have all the cells cleared the switch.

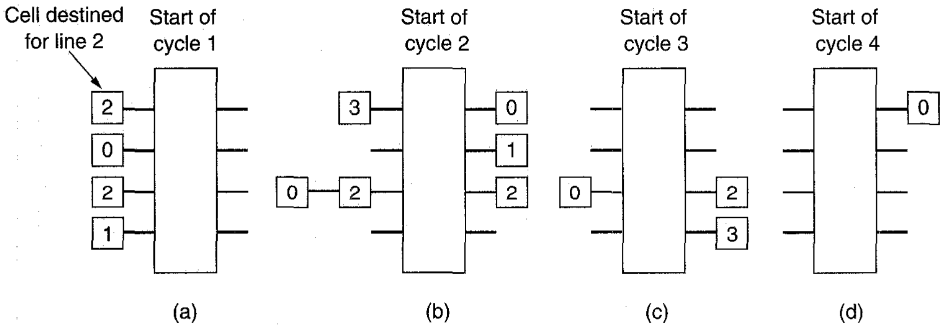


Fig. 2-46. Input queueing at an ATM switch.

The problem with input queueing is that when a cell has to be held up, it blocks the progress of any cells behind it, even if they could otherwise be switched. This effect is called **head-of-line blocking**. It is somewhat more complicated than shown here, since in a switch with 1024 input lines, conflicts may not be noticed until the cells are actually through the switch and fighting over the output line. Keeping a cell on its input queue until a signal comes back saying it made it through the switch requires extra logic, a reverse signaling path, and more delay. What is sometimes done is to put the losing cells on a recirculating bus that sends them back to the input side, but the switch has to be careful where it puts them, to avoid delivering cells from the same virtual circuit out of order.

An alternative design, one that does not exhibit head-of-line blocking does the queueing on the output side, as shown in Fig. 2-47. Here we have the same cell arrival pattern, but now when two cells want to go to the same output line in the same cycle, both are passed through the switch. One of them is put on the output line, and the other is queued on the output line, as shown in Fig. 2-47(b). Here it takes only three cycles, instead of four, to switch all the packets. Karol et al. (1987) have shown that output queueing is generally more efficient than input queueing.

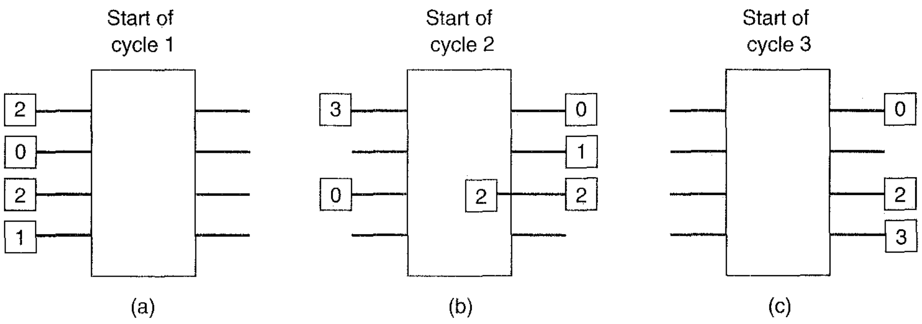


Fig. 2-47. Output queueing at an ATM switch.

The Knockout Switch

Let us now look more carefully at one ATM switch design that uses output queueing. It is called the **knockout switch** (Yeh et al., 1987) and is illustrated in Fig. 2-48 for eight input lines and eight output lines. Each input line is connected to a bus on which incoming cells are broadcast in the cycle they arrive. Having only one bus driver per bus simplifies the design and timing considerably.

For each arriving cell, hardware inspects the cell's header to find its virtual circuit information, looks that up in the routing tables (see Fig. 2-43), and enables the correct crosspoint. The cell then travels along its bus until it gets to the enabled crosspoint, at which time it heads south toward its output line. It is possible for multiple cells, in fact even all of them, to go to the same output line. It is also possible for a cell to be multicast to several output lines by just enabling several crosspoints on its broadcast bus.

The simplest way to handle collisions would be to simply buffer all cells at the output side. However, for a switch with 1024 input lines, in the worst case 1024 output buffers would be needed. In practice, this situation is very unlikely to occur, so a reasonable optimization is to provide far fewer output buffers, say *n*.

In the unlikely event that more cells arrive in one cycle than can be handled, the concentrator on each line selects out *n* cells for queueing, discarding the rest. The concentrator is a clever circuit for making this selection in a fair way, using

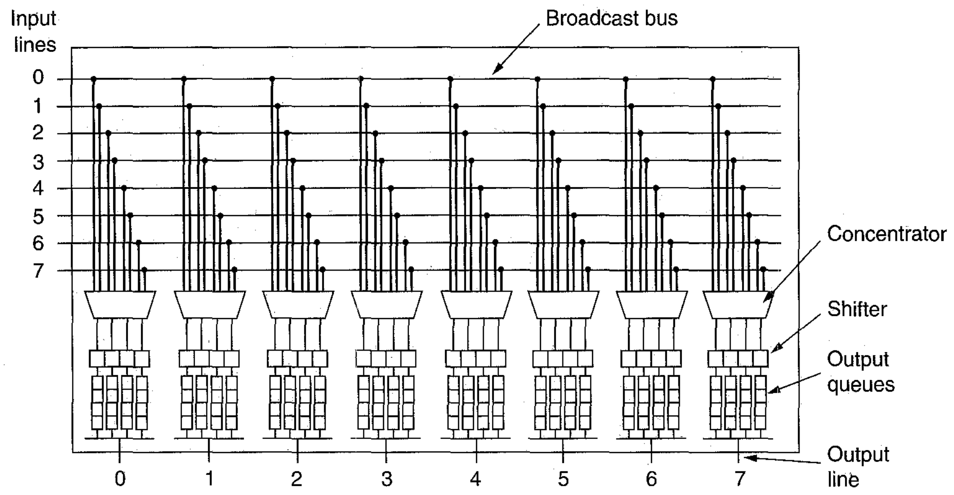


Fig. 2-48. A simplified diagram of the knockout switch.

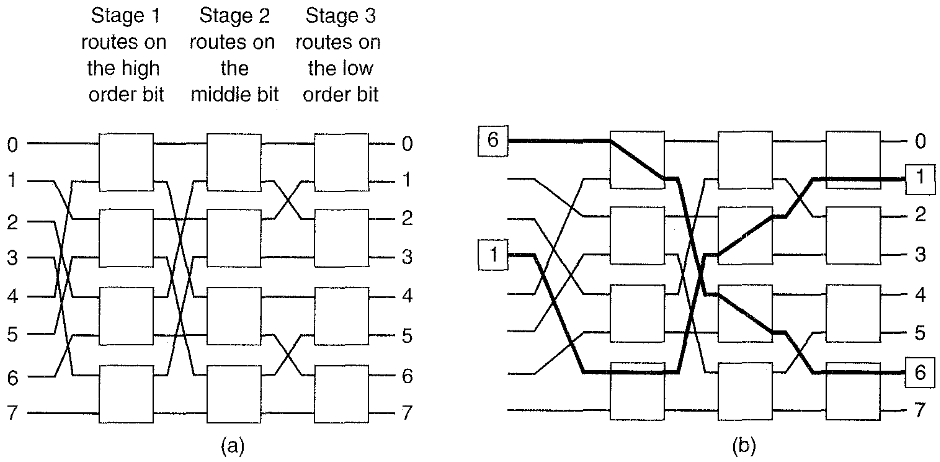
an elimination (knockout) tournament similar to the quarter finals, semifinals, and finals in many sports tournaments.

Conceptually, all the selected cells go into a single output queue (unless it is full, in which case cells are discarded). However, actually getting all the cells into a single queue in the allotted time is not feasible, so the output queue is simulated by multiple queues. The selected cells go into a shifter, which then distributes them uniformly over *n* output queues using a token to keep track of which queue goes next, in order to maintain sequencing within each virtual circuit. By varying *n*, the designers can trade switch cost off against expected cell loss rate.

The Batcher-Banyan Switch

The problem with the knockout switch is that it is basically a crossbar switch, so the number of crosspoints is quadratic in the number of lines. Just as this factor proved to be a problem with circuit switching, it is also a problem with packet switching. The solution for circuit switching was space division switching, which vastly reduced the number of crosspoints, at the cost of requiring a multistage switch. A similar solution is available for packet switching.

This solution is called the **Batcher-banyan switch**. Like knockout switches, Batcher-banyan switches are synchronous, accepting a set of cells (zero or one per input line) on each cycle. Even a simple Batcher-banyan is more complicated than the space division switches of Fig. 2-39, so we will introduce it step by step. In Fig. 2-49(a) we have an 8 × 8 three-stage banyan switch, so called because its wiring is said to resemble the roots of a banyan tree. In all banyan switches, only one path exists from each input line to each output line. Routing is done by looking up the output line for each cell (based on the virtual circuit information and tables). This 3-bit binary number is then put in front of the cell, as it will be used for routing through the switch.



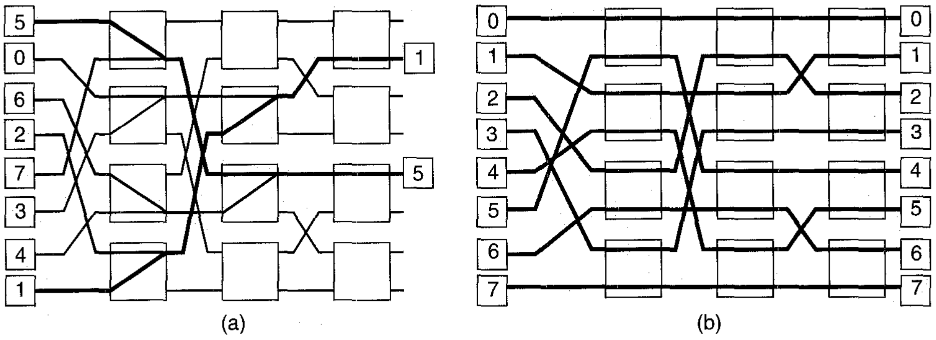
(a) (b)

Fig. 2-49. (a) A banyan switch with eight input lines and eight output lines. (b) The routes that two cells take through the banyan switch.

Each of the 12 switching elements in the banyan switch has two inputs and two outputs. When a cell arrives at a switching element, 1 bit of the output line number is inspected, and based on that, the cell is routed either to port 0 (the upper one) or port 1 (the lower one). In the event of a collision, one cell is routed and one is discarded.

A banyan switch parses the output line number from left to right, so stage 1 examines the leftmost (i.e., high-order) bit, stage 2 examines the middle bit, and stage 3 examines the rightmost (i.e., low-order) bit. In Fig. 2-49(b) we have two cells present: a cell on input line 0 heading for output line 6, and a cell on input line 3 heading for output line 1. For the first cell, the binary output address is 110, so it passes through the three stages using the lower, lower, and upper ports, respectively, as shown. Similarly, the other cell, labeled 001 in binary, uses the upper, upper, and lower ports, respectively.

Unfortunately, a collision occurs in a banyan switch when two incoming cells want to exit a switching element via the same port at the same time. A series of such collisions is illustrated in Fig. 2-50(a). In stage 1, the collisions involve the cells heading for the following pairs of output lines: (5, 7), (0, 3), (6, 4), and (2, 1). Suppose that these collisions are resolved in favor of 5, 0, 4, and 1. In the second stage we get collisions between (0, 1) and (5, 4). Here we let 1 and 5 win, and they are then routed to the correct output lines.



(a) (b)

Fig. 2-50. (a) Cells colliding in a banyan switch. (b) Collision-free routing through a banyan switch.

Now look at Fig. 2-50(b). All eight cells get through with no collisions. The conclusion is: depending on the input, the banyan switch can do a good job or a bad job of routing.

The idea behind the Batcher-banyan switch is to put a switch in front of the banyan switch to permute the cells into a configuration that the banyan switch can handle without loss. For example, if the incoming cells are sorted by destination and presented on input lines 0, 2, 4, 6, 1, 3, 5, and 7, in that order as far as necessary (depending on how many cells there are), then the banyan switch does not lose cells.

To sort the incoming cells we can use a Batcher switch, invented by K.E. Batcher (1968). Like the banyan and knockout switches, it too is synchronous and works with discrete cycles. A Batcher switch is built up of 2 × 2 switching elements, but these work differently than those in the banyan switch. When a switching element receives two cells, it compares their output addresses numerically (thus not just 1 bit) and routes the higher one on the port in the direction of the arrow, and the lower one the other way. If there is only one cell, it goes to the port opposite the way the arrow is pointing.

A Batcher switch for eight lines is depicted on the left in Fig. 2-51. Stage 1 sorts the incoming cells pairwise. The next two stages do a four-way merge. The final three stages do an eight-way merge. In general, for n lines, the complexity of a Batcher switch grows like $n\log^2 n$. When k cells are present on the input lines, the Batcher switch puts the cells in sort order on the first k output lines.

After exiting the Batcher switch, the cells undergo a shuffle and are then injected into a banyan switch. The final result is that every cell appears on the correct output line at the far end of the banyan switch.

An example of how the combined Batcher-banyan switching fabric works is given in Fig. 2-52. Here cells are present on input lines 2, 3, 4, and 5, headed for output lines 6, 5, 1, and 4, respectively. Initially, cells for 5 and 6 enter the same

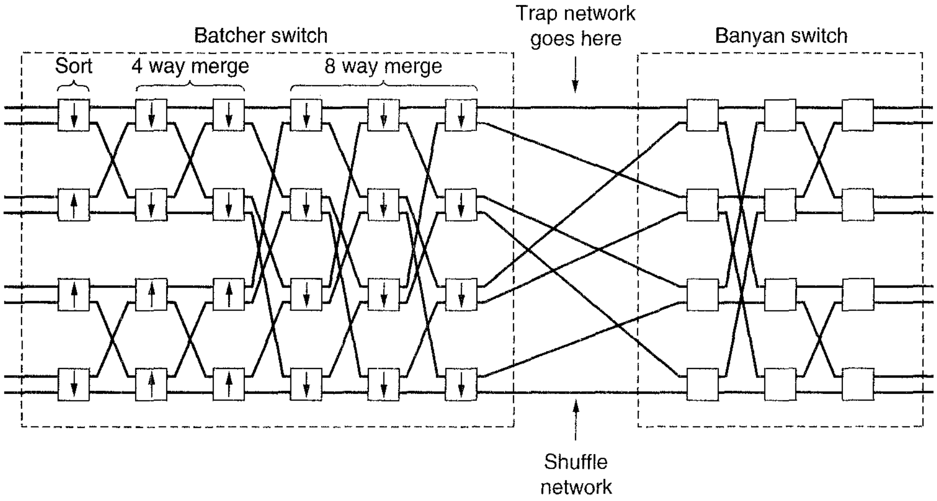


Fig. 2-51. The switching fabric for a Batcher-banyan switch.

switching element. Cell 6 has a higher address, so it exits in the direction of the arrow; cell 5 goes the other way. Here no exchange occurs. With cells 1 and 4, an exchange occurs, with cell 4 entering from the bottom but leaving from the top. The heavy lines show the paths all the way through to the end.

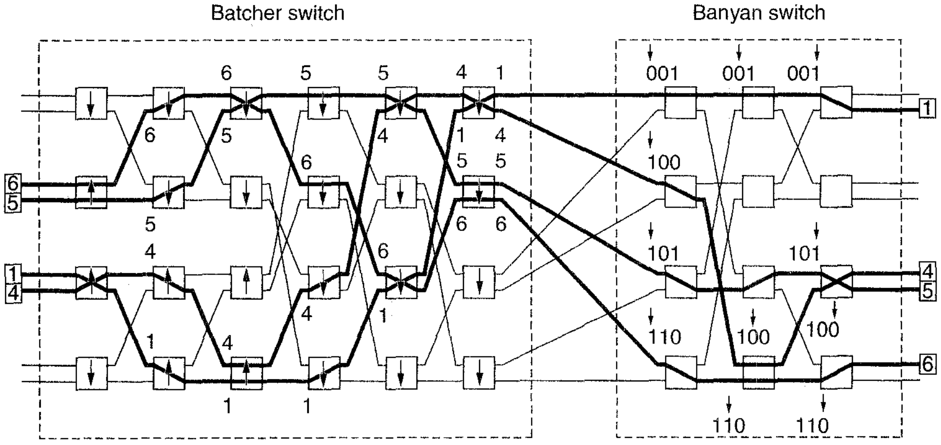


Fig. 2-52. An example with four cells using the Batcher-banyan switch.

Note that at the end of the Batcher switch, the four cells are stacked up at the top, in order. They are then run through a shuffle network and injected into the banyan switch, which is able to process them without collisions.

In principle, the Batcher-banyan switch makes a fine ATM switch, but there are two complications that we have ignored: output line collisions and multicasting. If two or more cells are aimed at the same output line, the Batcher-banyan switch cannot handle them, so we have to go back to some kind of buffering. One way to solve this problem is by inserting a trap network between the Batcher switch and the banyan switch. The job of the trap network is to filter out duplicates and recirculate them for subsequent cycles, all the while maintaining the order of cells on a virtual circuit. (By now it should be clear that the ordering requirement is much more of a problem than it might have at first appeared.) Commercial switches also have to handle multicast.

The first Batcher-banyan ATM switch was designed by Huang and Knauer (1984). It was called Starlite. Then came Moonshine (Hui, 1987) and Sunshine (Giacopelli et al., 1991). You have to admit that these folks have a sense of humor. Starlite, Moonshine, and Sunshine differ primarily in the trap circuit and how they handle multicast.

2.7. CELLULAR RADIO

The traditional telephone system (even when broadband ISDN is fully operational) will still not be able to satisfy a growing group of users: people on the go. Consequently, there is increasing competition from a system that uses radio waves instead of wires and fibers for communication. This system will play an increasingly important role in the networking of notebook computers, shirt-pocket telephones, and personal digital assistants in the coming years. In the following sections we will examine satellite paging, cordless telephones, cellular telephones, and similar technologies. These systems are now merging, producing portable computers capable of sending and receiving phone calls, faxes, and email, as well as looking up information in remote databases, and doing this anywhere on earth.

Such devices are already creating a huge market. Many companies in the computer, telephone, satellite, and other industries want a piece of the action. The result is a chaotic market, with numerous overlapping and incompatible products and services, all rapidly changing, and typically different in every country as well. Nevertheless, the descriptions given below should provide at least a basic knowledge of the underlying technologies. For more information, see (Bates, 1994; Goodman, 1991; Macario, 1993; Padgett et al., 1995; and Seybold, 1994).

2.7.1. Paging Systems

The first paging systems used loudspeakers within a single building. In a hospital it is common to hear announcements on the public address system like: Will Dr. Suzanne Johnson please call extension 4321? Nowadays, people who want to be paged wear small beepers, usually with tiny screens for displaying short incoming messages.

A person wanting to page a beeper wearer can then call the beeper company and enter a security code, the beeper number, and the number the beeper wearer is to call (or another short message). The computer receiving the request then transmits it over land lines to a hilltop antenna, which either broadcasts the page directly (for local paging), or sends it to an overhead satellite (for long-distance paging), which then rebroadcasts it. When the beeper detects its unique number in the incoming radio stream, it beeps and displays the number to be called. It is also possible to page a group of people simultaneously with a single phone call.

The most advanced beeper systems plug directly into a computer and can receive not just a telephone number, but a longer message. The computer can then process the data as they come in. For example, a company could keep the price lists in its salespeoples' portable computers up to date using this form of paging.

Most current paging systems have the property that they are one-way systems, from a single computer out to a large number of receivers. There is no problem about who will speak next, and no contention among many competing users for a small number of channels as there is only one sender in the whole system.

Paging systems require little bandwidth since each message requires only a single burst of perhaps 30 bytes. At this data rate, a 1-Mbps satellite channel can handle over 240,000 pages per minute. The older paging systems run at various frequencies in the 150–174 MHz band. Most of the modern ones run in the 930–932 MHz band. Figure 2-53(a) shows the one-way nature of a paging system, with all communication being outbound at a single frequency. We will later see how this mode contrasts with mobile telephones, which are two way and use two frequencies per call, with different frequency pairs used for different calls, as depicted in Fig. 2-53(b). These differences make the paging system much simpler and less expensive to operate.

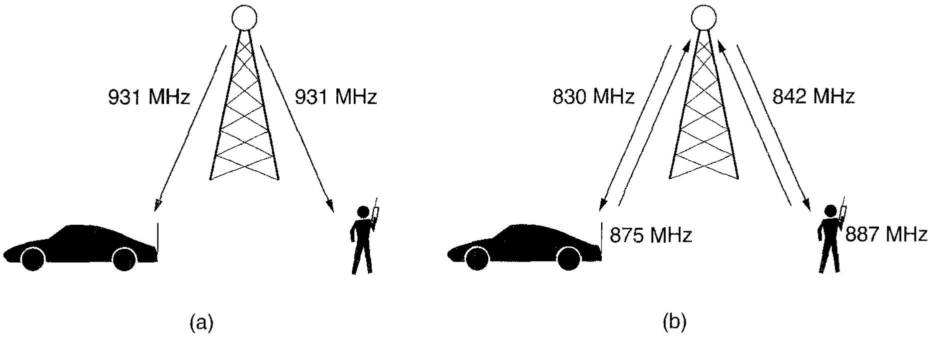


Fig. 2-53. (a) Paging systems are one way. (b) Mobile telephones are two way.

2.7.2. Cordless Telephones

Cordless telephones started as a way to allow people to walk around the house while on the phone. A cordless telephone consists of two parts: a base station and a telephone. These are always sold together. The base station has a standard phone jack at the back so it can be connected (by a wire) to the telephone system. The telephone communicates with the base station by low-power radio. The range is typically 100 to 300 meters.

Because early cordless telephones were only expected to communicate with their own base stations, there was no need for standardization. Some of the cheaper models used a fixed frequency, selected at the factory. If, by accident, your cordless phone happened to use the same frequency as your neighbor's, each of you could listen in on one another's calls. More expensive models avoided this problem by allowing the user to select the transmission frequency.

The first generation of cordless telephones, known as CT-1 in the United States and CEPT-1 in Europe, were entirely analog. They could, and often did, cause interference with radios and televisions. The poor reception and lack of security led the industry to develop a digital standard, CT-2, which originated in England. The first CT-2 devices could make calls, but not receive them, but as soon as the first one was sold, the manufacturer received some negative feedback and the system was quickly redesigned. Like the CT-1 version, each telephone had to be within a few hundred meters of its own base station, making it useful around the house or office, but useless in cars or when walking around town.

In 1992, a third generation, CT-3 or DECT, was introduced, which supported roaming over base stations. This technology is beginning to approach cellular telephones, which we will now describe.

2.7.3. Analog Cellular Telephones

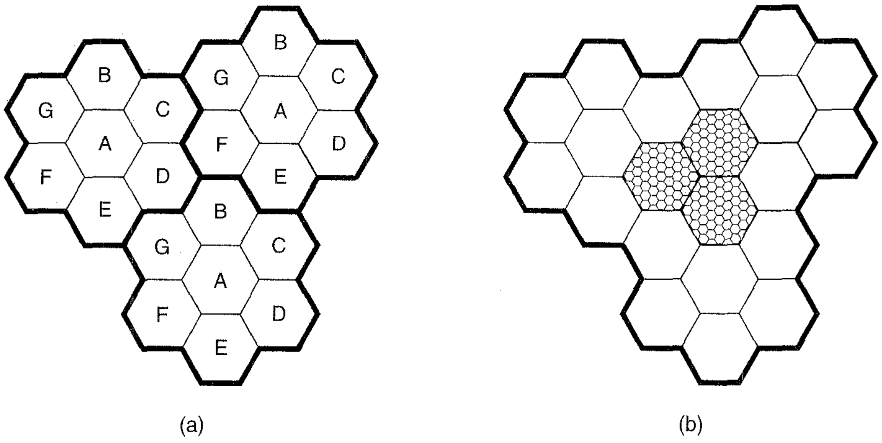
Mobile radiotelephones were used sporadically for maritime and military communication during the early decades of the 20th Century. In 1946, the first system for car-based telephones was set up in St. Louis. This system used a single large transmitter on top of a tall building and had a single channel, used for both sending and receiving. To talk, the user had to push a button that enabled the transmitter and disabled the receiver. Such systems, known as **push-to-talk systems**, were installed in several cities beginning in the late 1950s. CB-radio, taxis, and police cars on television programs often use this technology.

In the 1960s, **IMTS (Improved Mobile Telephone System)** was installed. It, too, used a high-powered (200-watt) transmitter, on top of a hill, but now had two frequencies, one for sending and one for receiving, so the push-to-talk button was no longer needed. Since all communication from the mobile telephones went inbound on a different channel than the telephones listened to, the mobile users could not hear each other (unlike the push-to-talk system used in taxis).

IMTS supported 23 channels spread out from 150 MHz to 450 MHz. Due to the small number of channels, users often had to wait a long time before getting a dial tone. Also, due to the large power of the hilltop transmitter, adjacent systems had to be several hundred kilometers apart to avoid interference. All in all, the system was impractical due to the limited capacity.

Advanced Mobile Phone System

All that changed with **AMPS** (**Advanced Mobile Phone System**), invented by Bell Labs and first installed in the United States in 1982. It is also used in England, where it is called TACS, and in Japan, where it is called MCS-L1. In AMPS, a geographic region is divided up into **cells**, typically 10 to 20 km across, each using some set of frequencies. The key idea that gives AMPS far more capacity than all previous systems is using relatively small cells, and reusing transmission frequencies in nearby (but not adjacent) cells. Whereas an IMTS system 100 km across can have one call on each frequency, an AMPS system might have 100 10-km cells in the same area and be able to have 5 to 10 calls on each frequency, in widely separated cells. Furthermore, smaller cells mean less power is needed, which leads to smaller and cheaper devices. Hand-held telephones put out 0.6 watts; transmitters in cars are typically 3 watts, the maximum allowed by the FCC.



(a) (b)

Fig. 2-54. (a) Frequencies are not reused in adjacent cells. (b) To add more users, smaller cells can be used.

The idea of frequency reuse is illustrated in Fig. 2-54(a). The cells are normally roughly circular, but they are easier to model as hexagons. In Fig. 2-54(a), the cells are all the same size. They are grouped together in units of seven cells. Each letter indicates a group of frequencies. Notice that for each frequency set, there is a buffer about two cells wide where that frequency is not reused, providing for good separation and low interference.

Finding locations high in the air to place base station antennas is a major issue. This problem has led some telecommunication carriers to forge alliances with the Roman Catholic Church, since the latter owns a substantial number of exalted potential antenna sites worldwide, all conveniently under a single management.

In an area where the number of users has grown to the point where the system is overloaded, the power is reduced and the overloaded cells are split into smaller cells to permit more frequency reuse, as shown in Fig. 2-54(b). How big the cells should be is a complex matter, and is treated in (Hac, 1995).

At the center of each cell is a base station to which all the telephones in the cell transmit. The base station consists of a computer and transmitter/receiver connected to an antenna. In a small system, all the base stations are connected to a single device called an **MTSO (Mobile Telephone Switching Office)** or **MSC (Mobile Switching Center)**. In a larger one, several MTSOs may be needed, all of which are connected to a second-level MTSO, and so on. The MTSOs are essentially end offices as in the telephone system, and are, in fact, connected to at least one telephone system end office. The MTSOs communicate with the base stations, each other, and the PSTN using a packet switching network.

At any instant, each mobile telephone is logically in one specific cell and under the control of that cell's base station. When a mobile telephone leaves a cell, its base station notices the telephone's signal fading away and asks all the surrounding base stations how much power they are getting from it. The base station then transfers ownership to the cell getting the strongest signal, that is, the cell where the telephone is now located. The telephone is then informed of its new boss, and if a call is in progress, it will be asked to switch to a new channel (because the old one is not reused in any of the adjacent cells). This process is called **handoff** and takes about 300 msec. Channel assignment is done by the MTSO, which is the nerve center of the system. The base stations are really just radio relays.

Channels

The AMPS system uses 832 full-duplex channels, each consisting of a pair of simplex channels. There are 832 simplex transmission channels from 824 to 849 MHz, and 832 simplex receive channels from 869 to 894 MHz. Each of these simplex channels is 30 kHz wide. Thus AMPS uses FDM to separate the channels.

In the 800-MHz band, radio waves are about 40 cm long and travel in straight lines. They are absorbed by trees and plants and bounce off the ground and buildings. It is possible that a signal sent by a mobile telephone will reach the base station by the direct path, but also slightly later after bouncing off the ground or a building. This may lead to an echo effect or signal distortion. Sometimes, it is even possible to hear a distant conversation that has bounced several times.

In the United States, the 832 channels in each city are allocated by the FCC. Of these, half are assigned to the local telephone company, the **wireline carrier** or **B-side carrier**. The other half are assigned to a new entrant in the cellular business, the **A-side carrier**. The idea is to make sure there are at least two competing cellular suppliers, to promote competition and lower prices.

However, the distinction between a telephone company and a cellular phone company is now blurred, since most telephone companies have a cellular partner, and in 1994 AT&T merged with McCaw Cellular, the largest cellular operator. It frequently occurs that a company is an A-side carrier in some markets and a B-side carrier in others. Additional mixing occurs because a carrier may sell or trade any or all of its 416 channel licenses.

The 832 channels are divided into four categories:

1. Control (base to mobile) to manage the system.
2. Paging (base to mobile) to alert mobile users to calls for them.
3. Access (bidirectional) for call setup and channel assignment.
4. Data (bidirectional) for voice, fax, or data.

Twenty-one of the channels are reserved for control, and these are wired into a PROM in each telephone. Since the same frequencies cannot be reused in nearby cells, the actual number of voice channels available per cell is much smaller than 832, typically about 45.

Call Management

Each mobile telephone in AMPS has a 32-bit serial number and 10-digit telephone number in its PROM. The telephone number is represented as a 3-digit area code, in 10 bits, and a 7-digit subscriber number, in 24 bits. When a phone is switched on, it scans a preprogrammed list of 21 control channels to find the most powerful signal. Mobile phones are preset to scan for A-side only, B-side only, A-side preferred, or B-side preferred, depending on which service(s) the customer has subscribed to. From the control channel, it learns the numbers of the paging and access channels.

The phone then broadcasts its 32-bit serial number and 34-bit telephone number. Like all the control information in AMPS, this packet is sent in digital form, multiple times, and with an error-correcting code, even though the voice channels themselves are analog.

When the base station hears the announcement, it tells the MTSO, which records the existence of its new customer and also informs the customer's home

MTSO of his current location. During normal operation, the mobile telephone reregisters about once every 15 minutes.

To make a call, a mobile user switches on the phone, enters the number to be called on the keypad, and hits the SEND button. The phone then sends the number to be called and its own identity on the access channel. If a collision occurs there, it tries again later. When the base station gets the request, it informs the MTSO. If the caller is a customer of the MTSO's company (or one of its partners), the MTSO looks for an idle channel for the call. If one is found, the channel number is sent back on the control channel. The mobile phone then automatically switches to the selected voice channel and waits until the called party picks up the phone.

Incoming calls work differently. To start with, all idle phones continuously listen to the paging channel to detect messages directed at them. When a call is placed to a mobile phone (either from a fixed phone or another mobile phone), a packet is sent to the callee's home MTSO to find out where it is. A packet is then sent to the base station in its current cell, which then sends a broadcast on the paging channel of the form: "Unit 14, are you there?" The called phone then responds with "Yes" on the control channel. The base then says something like: "Unit 14, call for you on channel 3." At this point, the called phone switches to channel 3 and starts making ringing sounds.

Security Issues

Analog cellular phones are totally insecure. Anyone with an all-band radio receiver (scanner) can tune in and hear everything going on in a cell. Princess Di and her lover were once caught this way, which resulted in worldwide headlines. Since most cellular users do not realize how insecure the system is, they often give out credit card numbers and other once-confidential information this way.

Another major problem is theft of air time. With an all-band receiver attached to a computer, a thief can monitor the control channel and record the 32-bit serial number and 34-bit telephone numbers of all the mobile telephones it hears. By just driving around for a couple of hours, he can build up a large database. The thief can then pick a number and use it for his calls. This trick will work until the victim gets the bill, weeks later, at which time the thief just picks a new number.

Some thieves offer a low-cost telephone service by making calls for their customers using stolen numbers. Others reprogram mobile telephones with stolen numbers and sell them as phones that can make free calls.

Some of these problems could be solved by encryption, but then the police could not easily perform "wiretaps" on wireless criminals. This subject is very controversial and is discussed in more detail in Chap. 7.

Another issue in the general area of security is vandalism and damage to antennas and base stations. All these problems are quite severe and add up to hundreds of millions of dollars a year in losses for the cellular industry.

2.7.4. Digital Cellular Telephones

First generation cellular systems were analog. The second generation is digital. In the United States there was basically only one system: AMPS. When it was time for digital, three or four competitors emerged, and a struggle for survival began. It now appears that two systems will survive. The first one is backward compatible with the AMPS frequency allocation scheme and is specified in standards known as IS-54 and IS-135. The other is based on direct sequence spread spectrum and is specified in standard IS-95.

IS-54 is dual mode (analog and digital) and uses the same 30-kHz channels that AMPs does. It packs 48.6 kbps in each channel and shares it among three simultaneous users. Each user gets 13 kbps; the rest is control and timing overhead. Cells, base stations, and MTSOs work the same as in AMPS. Only the digital signaling and digital voice encoding is different. The IS-95 system is quite novel. We will discuss it when we get to channel allocation in Chap. 4.

In Europe, the reverse process happened. Five different analog systems were in use, in different countries, so someone with a British phone could not use it in France, and so on. This experience led the European PTTs to agree on a common digital system, called **GSM** (**Global Systems for Mobile communications**), which was deployed before any of the competing American systems. The Japanese system is different from all of the above.

Since the European systems were all different, it was simplest to make them pure digital operating in a new frequency band (1.8 GHz), in addition to retrofitting the 900-MHz band where possible. GSM uses both FDM and TDM. The available spectrum is broken up into 50 200-kHz bands. Within each band TDM is used to multiplex multiple users.

Some GSM telephones use smart cards, that is, credit card sized devices containing a CPU. The serial number and telephone number are contained there, not in the telephone, making for better physical security (stealing the phone without the card will not get you the number). Encryption is also used. We will discuss GSM in Chap. 4.

2.7.5. Personal Communications Services

The holy grail of the telephone world is a small cordless phone that you can use around the house and take with you anywhere in the world. It should respond to the same telephone number, no matter where it is, so people only have one telephone number (with AMPS, your home phone and your mobile phone have different numbers). This system is currently under vigorous development (Lipper and Rumsewicz, 1994). In the United States it is called **PCS** (**Personal Communications Services**). Everywhere else it is called **PCN** (**Personal Communications Network**). In the world of telephony, the United States has something of a tradition of marching to a different drummer than everyone else. Fortunately, most of the technical details are the same.

PCS will use cellular technology, but with microcells, perhaps 50 to 100 meters wide. This allows very low power (1/4 watt), which makes it possible to build very small, light phones. On the other hand, it requires many more cells than the 20-km AMPS cells. If we assume that a microcell is 1/200th the diameter of an AMPS cell, 40,000 times as many cells are required to cover the same area. Even if these microcells are much cheaper than AMPS cells, it is clear that building a complete PCS system from scratch will require a far more massive investment in infrastructure than did AMPS. Some telephone companies have realized that their telephone poles are excellent places to put the toaster-sized base stations, since the poles and wires already exist, thus greatly reducing the installation costs. These small base stations are sometimes called **telepoints**. How many to install and where to put them is a complicated issue (Steele et al., 1995a, 1995b).

The U.S. government (specifically, the FCC) is using PCS to make money out of thin air. In 1994–95 it auctioned off licenses to use the PCS spectrum (1.7 to 2.3 GHz). The auction raised 7.7 billion dollars for the government. This auction replaced the previous system of awarding frequency bands by lottery, thus eliminating the practice of companies with no interest in telecommunication entering the lottery. Any such company winning a frequency could instantly sell it to one of the losers for millions of dollars.

Unfortunately, there is no such thing as a free lunch, not even for the government. The 1.7- to 2.3-GHz band is already completely allocated to other users. These users will be given spectrum elsewhere and told to move there. The trouble is, antenna size depends on frequency, so this forced frequency reallocation will require a multibillion dollar investment in antennas, transmitters, etc. to be thrown away. Hordes of lobbyists are roaming around Washington with suggestions as to who should pay for all this. The net result is that PCS may not be widely deployed in this millenium. For a more rational way to deal with the spectrum, see (Youssef et al., 1995).

2.8. COMMUNICATION SATELLITES

In the 1950s and early 1960s, people tried to set up communication systems by bouncing signals off metallized weather balloons. Unfortunately, the received signals were too weak to be of any practical use. Then the U.S. Navy noticed a kind of permanent weather balloon in the sky—the moon—and built an operational system for ship-to-shore communication by bouncing signals off it.

Further progress in the celestial communication field had to wait until the first communication satellite was launched in 1962. The key difference between an artificial satellite and a real one is that the artificial one can amplify the signals before sending them back, turning a strange curiosity into a powerful communication system.

Communication satellites have some interesting properties that make them attractive for many applications. A communication satellite can be thought of as a big microwave repeater in the sky. It contains several **transponders**, each of which listens to some portion of the spectrum, amplifies the incoming signal, and then rebroadcasts it at another frequency, to avoid interference with the incoming signal. The downward beams can be broad, covering a substantial fraction of the earth's surface, or narrow, covering an area only hundreds of kilometers in diameter.

2.8.1. Geosynchronous Satellites

According to Kepler's law, the orbital period of a satellite varies as the orbital radius to the 3/2 power. Near the surface of the earth, the period is about 90 min. Communication satellites at such low altitudes are problematic because they are within sight of any given ground station for only a short time interval.

However, at an altitude of approximately 36,000 km above the equator, the satellite period is 24[†] hours, so it revolves at the same rate as the earth under it. An observer looking at a satellite in a circular equatorial orbit sees the satellite hang in a fixed spot in the sky, apparently motionless. Having the satellite be fixed in the sky is extremely desirable, because otherwise an expensive steerable antenna would be needed to track it.

With current technology, it is unwise to have satellites spaced much closer than 2 degrees in the 360-degree equatorial plane, to avoid interference. With a spacing of 2 degrees, there can only be 360/2 = 180 geosynchronous communication satellites in the sky at once. Some of these orbit slots are reserved for other classes of users (e.g., television broadcasting, government and military use, etc.).

Fortunately, satellites using different parts of the spectrum do not compete, so each of the 180 possible satellites could have several data streams going up and down simultaneously. Alternatively, two or more satellites could occupy one orbit slot if they operate at different frequencies.

To prevent total chaos in the sky, there have been international agreements about who may use which orbit slots and frequencies. The main commercial bands are listed in Fig. 2-55. The C band was the first to be designated for commercial satellite traffic. Two frequency ranges are assigned in it, the lower one for downlink traffic (from the satellite) and the upper one for uplink traffic (to the satellite). For a full-duplex connection one channel each way is required. These bands are already overcrowded because they are also used by the common carriers for terrestrial microwave links.

The next highest band available to commercial telecommunication carriers is

---

[†] For the purist, the rotation rate is the sidereal day: 23 hours 56 minutes 4.09 seconds.

Band	Frequencies	Downlink (GHz)	Uplink (GHz)	Problems
C	4/6	3.7–4.2	5.925–6.425	Terrestrial interference
Ku	11/14	11.7–12.2	14.0–14.5	Rain
Ka	20/30	17.7–21.7	27.5–30.5	Rain; equipment cost

Fig. 2-55. The principal satellite bands.

the Ku band. This band is not (yet) congested, and at these frequencies satellites can be spaced as close as 1 degree. However, another problem exists: rain. Water is an excellent absorber of these short microwaves. Fortunately, heavy storms are usually localized, so by using several widely separated ground stations instead of just one, the problem can be circumvented at the price of extra antennas, extra cables, and extra electronics to switch rapidly between stations. Bandwidth has also been allocated in the Ka band for commercial satellite traffic, but the equipment needed to use them is still expensive. In addition to these commercial bands, many government and military bands also exist.

A typical satellite has 12–20 transponders, each with a 36–50-MHz bandwidth. A 50-Mbps transponder can be used to encode a single 50-Mbps data stream, 800 64-kbps digital voice channels, or various other combinations. Furthermore, two transponders can use different polarizations of the signal, so they can use the same frequency range without interfering. In the earliest satellites, the division of the transponders into channels was static, by splitting the bandwidth up into fixed frequency bands (FDM). Nowadays, time division multiplexing is also used due to its greater flexibility.

The first satellites had a single spatial beam that illuminated the entire earth. With the enormous decline in the price, size, and power requirements of microelectronics, a much more sophisticated broadcasting strategy has become possible. Each satellite is equipped with multiple antennas and multiple transponders. Each downward beam can be focused on a small geographical area, so multiple upward and downward transmissions can take place simultaneously. These so-called **spot beams** are typically elliptically shaped, and can be as small as a few hundred km in diameter. A communication satellite for the United States would typically have one wide beam for the contiguous 48 states, plus spot beams for Alaska and Hawaii.

A new development in the communication satellite world is the development of low-cost microstations, sometimes called **VSATs** (**Very Small Aperture Terminals**) (Ivancic et al., 1994). These tiny terminals have 1-meter antennas and can put out about 1 watt of power. The uplink is generally good for 19.2 kbps, but the downlink is more, often 512 kbps. In many VSAT systems, the microstations do not have enough power to communicate directly with one another (via the satellite, of course). Instead, a special ground station, the **hub**, with a large, high-gain antenna is needed to relay traffic between VSATs, as shown in Fig. 2-56. In this mode of operation, either the sender or the receiver has a large antenna and a powerful amplifier. The trade-off is a longer delay in return for having cheaper end-user stations.

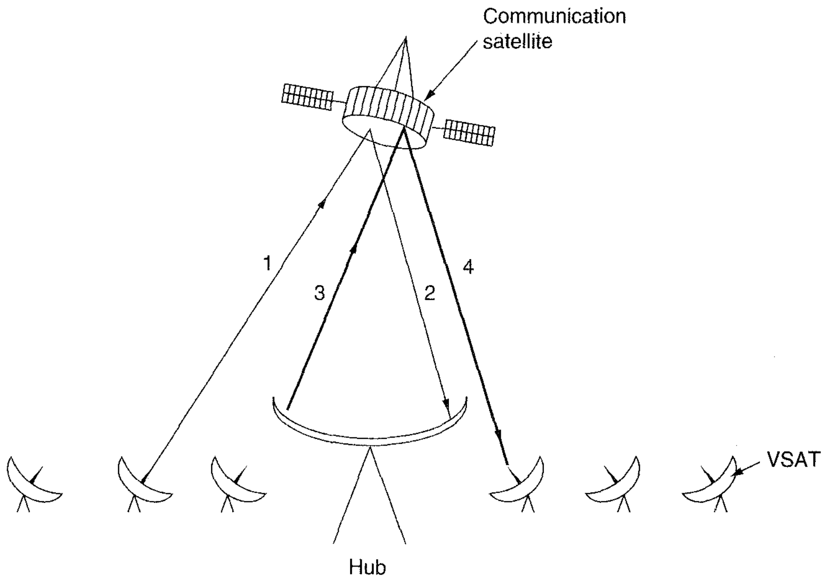


Fig. 2-56. VSATs using a hub.

Communication satellites have several properties that are radically different from terrestrial point-to-point links. To begin with, even though signals to and from a satellite travel at the speed of light (nearly 300,000 km/sec), the large round-trip distance introduces a substantial delay. Depending on the distance between the user and the ground station, and the elevation of the satellite above the horizon, the end-to-end transit time is between 250 and 300 msec. A typical value is 270 msec (540 msec for a VSAT system with a hub).

For comparison purposes, terrestrial microwave links have a propagation delay of roughly 3 μsec/km and coaxial cable or fiber optic links have a delay of approximately 5 μsec/km (electromagnetic signals travel faster in air than in solid materials).

Another important property of satellites is that they are inherently broadcast media. It does not cost more to send a message to thousands of stations within a transponder's footprint than it does to one. For some applications, this property is very useful. Even when broadcasting can be simulated using point-to-point line, satellite broadcasting may be much cheaper. On the other hand, from a security and privacy point of view, satellites are a complete disaster: everybody can hear everything. Encryption is essential when security is required.

Satellites also have the property that the cost of transmitting a message is independent of the distance traversed. A call across the ocean costs no more to service than a call across the street. Satellites also have excellent error rates and can be deployed almost instantly, a major consideration for military communication.

2.8.2. Low-Orbit Satellites

For the first 30 years of the satellite era, low-orbit satellites were rarely used for communication because they zip into and out of view so quickly. In 1990, Motorola broke new ground by filing an application with the FCC asking for permission to launch 77 low-orbit satellites for the Iridium project (element 77 is iridium). The plan was later revised to use only 66 satellites, so the project should have been renamed Dysprosium (element 66), but that probably sounded too much like a disease. The idea was that as soon as one satellite went out of view, another would replace it. This proposal set off a feeding frenzy among other communication companies. All of a sudden, everyone wanted to launch a chain of low-orbit satellites. We will briefly describe the Iridium system here, but the others are similar.

The basic goal of Iridium is to provide worldwide telecommunication service using hand-held devices that communicate directly with the Iridium satellites. It provides voice, data, paging, fax, and navigation service everywhere on earth. This service competes head-on with PCS/PCN and makes the latter unnecessary.

It uses ideas from cellular radio, but with a twist. Normally, the cells are fixed, but the users are mobile. Here, each satellite has a substantial number of spot beams that scan the earth as the satellite moves. Thus both the cells and the users are mobile in this system, but the handover techniques used for cellular radio are equally applicable to the case of the cell leaving the user as to the case of the user leaving the cell.

The satellites are to be positioned at an altitude of 750 km, in circular polar orbits. They would be arranged in north-south necklaces, with one satellite every 32 degrees of latitude. With six satellite necklaces, the entire earth would be covered, as suggested by Fig. 2-57(a). People not knowing much about chemistry can think of this arrangement as a very, very big dysprosium atom, with the earth as the nucleus and the satellites as the electrons.

Each satellite would have a maximum of 48 spot beams, with a total of 1628 cells over the surface of the earth, as shown in Fig. 2-57(b). Frequencies could be reused two cells away, as with conventional cellular radio. Each cell would have 174 full-duplex channels, for a total of 283,272 channels worldwide. Some of these would be for paging and navigation, which require hardly any bandwidth at all. (The paging devices envisioned would display two lines of alphanumeric text.)

The uplinks and downlinks would operate in the L band, at 1.6 GHz, making

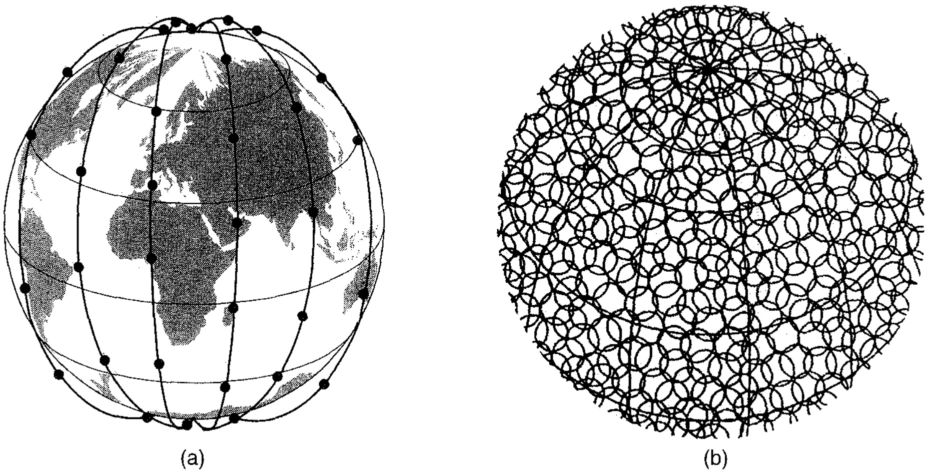


Fig. 2-57. (a) The Iridium satellites form six necklaces around the earth. (b) 1628 moving cells cover the earth.

it possible to communicate with a satellite using a small battery-powered device. Messages received by one satellite but destined for a remote one would be relayed among the satellites in the Ka band. Sufficient bandwidth is available in outer space for the intersatellite links. The limiting factor would be the uplink/downlink segments. Motorola estimates that 200 MHz would be sufficient for the whole system.

The projected cost to the end user is about 3 dollars per minute. If this technology can provide universal service anywhere on earth for that price, it is unlikely that the project will die for lack of customers. Business and other travelers who want to be in touch all the time, even in undeveloped areas, will sign up in droves. However, in developed areas, Iridium will face stiff competition from PCS/PCN with their toaster-on-a-pole telepoints.

2.8.3. Satellites versus Fiber

A comparison between satellite communication and terrestrial communication is instructive. As recently as 20 years ago, a case could be made that the future of communication lay with communication satellites. After all, the telephone system had changed little in the past 100 years and showed no signs of changing in the next 100 years. This glacial movement was caused in no small part by the regulatory environment in which the telephone companies were expected to provide good voice service at reasonable prices (which they did), and in return got a guaranteed profit on their investment. For people with data to transmit, 1200-bps modems were available. That was pretty much all there was.

The introduction of competition in 1984 in the United States and somewhat later in Europe changed all that radically. Telephone companies began replacing their long-haul networks with fiber and introduced high-bandwidth services like SMDS and B-ISDN. They also stopped their long-time practice of charging artificially high prices to long-distance users to subsidize local service.

All of a sudden, terrestrial fiber connections looked like the long-term winner. Nevertheless, communication satellites have some major niche markets that fiber does not (and sometimes, cannot) address. We will now look at a few of these.

While a single fiber has, in principle, more potential bandwidth than all the satellites ever launched, this bandwidth is not available to most users. The fibers that are now being installed are used within the telephone system to handle many long distance calls at once, not to provide individual users with high bandwidth. Furthermore, few users even have access to a fiber channel because the trusty old twisted pair local loop is in the way. Calling up the local telephone company end office at 28.8 kbps will never give more bandwidth than 28.8 kbps, no matter how wide the intermediate link is. With satellites, it is practical for a user to erect an antenna on the roof of the building and completely bypass the telephone system. For many users, bypassing the local loop is a substantial motivation.

For users who (sometimes) need 40 or 50 Mbps, an option is leasing a (44.736-Mbps) T3 carrier. However, this is an expensive undertaking. If that bandwidth is only needed intermittently, SMDS may be a suitable solution, but it is not available everywhere, and satellite service is.

A second niche is for mobile communication. Many people nowadays want to communicate while jogging, driving, sailing, and flying. Terrestrial fiber optic links are of no use to them, but satellite links potentially are. It is possible, however, that a combination of cellular radio and fiber will do an adequate job for most users (but probably not for those airborne or at sea).

A third niche is for situations in which broadcasting is essential. A message sent by satellite can be received by thousands of ground stations at once. For example, an organization transmitting a stream of stock, bond, or commodity prices to thousands of dealers might find a satellite system much cheaper than simulating broadcasting on the ground.

A fourth niche is for communication in places with hostile terrain or a poorly developed terrestrial infrastructure. Indonesia, for example, has its own satellite for domestic telephone traffic. Launching one satellite was much easier than stringing thousands of undersea cables among all the islands in the archipelago.

A fifth niche market for satellites is where obtaining the right of way for laying fiber is difficult or unduly expensive. Sixth, when rapid deployment is critical, as in military communication systems in time of war, satellites win easily.

In short, it looks like the mainstream communication of the future will be terrestrial fiber optics combined with cellular radio, but for some specialized uses, satellites are better. However, there is one caveat that applies to all of this: economics. Although fiber offers more bandwidth, it is certainly possible that terrestrial and satellite communication will compete aggressively on price. If advances in technology radically reduce the cost of deploying a satellite (e.g., some future space shuttle can toss out dozens of satellites on one launch), or low-orbit satellites catch on, it is not certain that fiber will win in all markets.

2.9. SUMMARY

The physical layer is the basis of all networks. Nature imposes two fundamental limits on al channels, and these determine their bandwidth. These limits are the Nyquist limit, which deals with noiseless channels, and the Shannon limit, for noisy channels.

Transmission media can be guided or unguided. The principle guided media are twisted pair, coaxial cable, and fiber optics. Unguided media include radio, microwaves, infrared, and lasers through the air.

A key element in most wide area networks is the telephone system. Its main components are the local loops, trunks, and switches. Local loops are analog, twisted pair circuits, which require modems for transmitting digital data. Trunks are digital, and can be multiplexed in several ways, including FDM, TDM, and WDM. The switches include crossbars, space division switches, and time division switches. Both circuit switching and packet switching are important.

In the future, the telephone system will be digital from end to end and will carry both voice and nonvoice traffic over the same lines. Two variants of this new system, known as ISDN, are being introduced. Narrowband ISDN is a circuit-switched digital system that is an incremental improvement over the current system. In contrast, broadband ISDN represents a paradigm shift, since it is based on cell switching ATM technology. Various kinds of ATM switches exist, including the knockout switch and the Batcher-banyan switch.

For mobile applications, the hard-wired telephone system is not suitable. Alternatives to the telephone system include cellular radio and communication satellites. Cellular radio is now widely used for portable telephones but will soon be common for data traffic as well. The current generation of cellular systems (e.g., AMPS) are analog, but the next generation (e.g., PCS/PCN) will be fully digital. Traditional communication satellites are geosynchronous, but there is now much interest in low-orbit satellite systems such as Iridium.

PROBLEMS

1. Compute the Fourier coefficients for the function $f(t) = t \quad (0 \le t \le 1)$.
2. A noiseless 4-kHz channel is sampled every 1 msec. What is the maximum data rate?
3. Television channels are 6 MHz wide. How many bits/sec can be sent if four-level digital signals are used? Assume a noiseless channel.

4. If a binary signal is sent over a 3-kHz channel whose signal-to-noise ratio is 20 dB, what is the maximum achievable data rate?

5. What signal-to-noise ratio is needed to put a T1 carrier on a 50-kHz line?

6. What is the difference between a passive star and an active repeater in a fiber optic network?

7. How much bandwidth is there in 0.1 micron of spectrum at a wavelength of 1 micron?

8. It is desired to send a sequence of computer screen images over an optical fiber. The screen is 480×640 pixels, each pixel being 24 bits. There are 60 screen images per second. How much bandwidth is needed, and how many microns of wavelength are needed for this band at 1.30 microns?

9. Is the Nyquist theorem true for optical fiber, or only for copper wire?

10. In Fig. 2-6 the lefthand band is narrower than the others. Why?

11. Radio antennas often work best when the diameter of the antenna is equal to the wavelength of the radio wave. Reasonable antennas range from 1 cm to 5 meters in diameter. What frequency range does this cover?

12. Multipath fading is maximized when the two beams arrive 180 degrees out of phase. How much of a path difference is required to maximize the fading for a 50 km long 1 GHz microwave link?

13. A laser beam 1 mm wide is aimed at a detector 1 mm wide 100 m away on the roof of a building. How much of an angular diversion (in degrees) does the laser have to have before it misses the detector?

14. A simple telephone system consists of two end offices and a single toll office to which each end office is connected by a 1-MHz full-duplex trunk. The average telephone is used to make four calls per 8-hour workday. The mean call duration is 6 min. Ten percent of the calls are long-distance (i.e., pass through the toll office). What is the maximum number of telephones an end office can support? (Assume 4 kHz per circuit.)

15. A regional telephone company has 10 million subscribers. Each of their telephones is connected to a central office by a copper twisted pair. The average length of these twisted pairs is 10 km. How much is the copper in the local loops worth? Assume that the cross section of each strand is a circle 1 mm in diameter, the specific gravity of copper is 9.0, and that copper sells for 3 dollars per kilogram.

16. The cost of a powerful microprocessor has dropped to the point where it is now possible to include one in each modem. How does that affect the handling of telephone line errors?

17. A modem constellation diagram similar to Fig. 2-19 has data points at the following coordinates: $(1, 1)$, $(1, -1)$, $(-1, 1)$, and $(-1, -1)$. How many bps can a modem with these parameters achieve at 1200 baud?

18. A modem constellation diagram similar to Fig. 2-19 has data points at $(0, 1)$ and $(0, 2)$. Does the modem use phase modulation or amplitude modulation?

19. Does FTTH fit into the telephone company model of end offices, toll offices, and so on, or does the model have to be changed in a fundamental way? Explain your answer.

20. At the low end, the telephone system is star shaped, with all the local loops in a neighborhood converging on an end office. In contrast, cable television consists of a single long cable snaking its way past all the houses in the same neighborhood. Suppose that a future TV cable were 10 Gbps fiber instead of copper. Could it be used to simulate the telephone model of everybody having their own private line to the end office? If so, how many one-telephone houses could be hooked up to a single fiber?

21. A cable TV system has 100 commercial channels, all of them alternating programs with advertising. Is this more like TDM or like FDM?

22. Why has the PCM sampling time been set at 125 μsec?

23. What is the percent overhead on a T1 carrier; that is, what percent of the 1.544 Mbps are not delivered to the end user?

24. Compare the maximum data rate of a noiseless 4-kHz channel using
(a) Analog encoding with 2 bits per sample.
(b) The T1 PCM system.

25. If a T1 carrier system slips and loses track of where it is, it tries to resynchronize using the 1st bit in each frame. How many frames will have to be inspected on the average to resynchronize with a probability of 0.001 of being wrong?

26. What is the difference, if any, between the demodulator part of a modem and the coder part of a codec? (After all, both convert analog signals to digital ones.)

27. A signal is transmitted digitally over a 4-kHz noiseless channel with one sample every 125 μsec. How many bits per second are actually sent for each of these encoding methods?
(a) CCITT 2.048 Mbps standard.
(b) DPCM with a 4-bit relative signal value.
(c) Delta modulation.

28. A pure sine wave of amplitude A is encoded using delta modulation, with x samples/sec. An output of +1 corresponds to a signal change of $+A/8$, and an output signal of −1 corresponds to a signal change of $-A/8$. What is the highest frequency that can be tracked without cumulative error?

29. SONET clocks have a drift rate of about 1 part in 10^9. How long does it take for the drift to equal the width of 1 bit? What are the implications of this calculation?

30. In Fig. 2-32, the user data rate for OC-3 is stated to be 148.608 Mbps. Show how this number can be derived from the SONET OC-3 parameters.

31. What is the available user bandwidth in an OC-12c connection?

32. Three packet-switching networks each contain n nodes. The first network has a star topology with a central switch, the second is a (bidirectional) ring, and the third is fully interconnected, with a wire from every node to every other node. What are the best, average, and worst case transmission paths in hops?

33. Compare the delay in sending an x-bit message over a k-hop path in a circuit-switched network and in a (lightly loaded) packet-switched network. The circuit setup time is s sec, the propagation delay is d sec per hop, the packet size is p bits, and the data rate is b bps. Under what conditions does the packet network have a lower delay?

34. Suppose that x bits of user data are to be transmitted over a k-hop path in a packet-switched network as a series of packets, each containing p data bits and h header bits, with $x \gg p + h$. The bit rate of the lines is b bps and the propagation delay is negligible. What value of p minimizes the total delay?

35. How many crosspoints do the switches of Fig. 2-39(a) and Fig. 2-39(b) have? Compare this to a full 16×16 single-stage crossbar switch.

36. In the space division switch of Fig. 2-39(a), what is the smallest number of existing connections that can block a new outgoing call?

37. An alternative design to that of Fig. 2-39(a) is one in which the 16 lines are divided into two blocks of eight, instead of four blocks of four (i.e., $n = 8$ instead of $n = 4$). Such a design would save on hardware costs, since only two concentrators would be needed on the input and output sides. What is the strongest argument against this alternative?

38. How many lines can a time division switch handle if the RAM access time is 50 nsec?

39. How many bits of RAM buffer does a time switch interchanger need if the input line samples are 10 bits and there are 80 input lines?

40. Does time division switching necessarily introduce a minimum delay at each switching stage? If so, what is it?

41. How long does it take to transmit an 8 inch by 10 inch image by facsimile over an ISDN B channel? The facsimile digitizes the image into 300 pixels per inch and assigns 4 bits per pixel. Current FAX machines go faster than this over ordinary telephone lines. How do you think they do it?

42. Give an advantage and a disadvantage of NT12 (as opposed to NT1 and NT2) in an ISDN network.

43. In Fig. 2-50(a) we saw collisions between cells traveling through a banyan switch. These collisions occurred in the first and second stages. Can collisions also occur in the third stage? If so, under what conditions?

44. For this problem you are to route some cells through a Batcher-banyan ATM switch step by step. Four cells are present on input lines 0 through 3, headed for 3, 5, 2, and 1 respectively. For each of the six stages in the Batcher switch and the four steps in the banyan switch (including the input and output), list which cells are there as an eight-tuple (cell on line 0, cell on line 1, and so on). Indicate lines with no cell by –.

45. Now repeat the previous problem starting from (7, –, 6, –, 5, – 4, –).

46. An ATM switch has 1024 input lines and 1024 output lines. The lines operate at the SONET rate of 622 Mbps, which gives a user rate of approximately 594 Mbps. What aggregate bandwidth does the switch need to handle the load? How many cells per second must it be able to process?

47. In a typical cellular telephone system with hexagonal cells, it is forbidden to reuse a frequency band in an adjacent cell. If a total of 840 frequencies are available, how many can be used in a given cell?

48. Make a rough estimate of the number of PCS microcells 100 m in diameter it would take to cover San Francisco (120 square km).

49. Sometimes when a cellular user crosses the boundary from one cell to another, the current call is abruptly terminated, even though all transmitters and receivers are functioning perfectly. Why?

50. The 66 low-orbit satellites in the Iridium project are divided into six necklaces around the earth. At the altitude they are using, the period is 90 minutes. What is the average interval for handoffs for a stationary transmitter?

3

THE DATA LINK LAYER

In this chapter we will study the design of layer 2, the data link layer. This study deals with the algorithms for achieving reliable, efficient communication between two adjacent machines at the data link layer. By adjacent, we mean that the two machines are physically connected by a communication channel that acts conceptually like a wire (e.g., a coaxial cable or a telephone line). The essential property of a channel that makes it "wire-like" is that the bits are delivered in exactly the same order in which they are sent.

At first you might think this problem is so trivial that there is no software to study—machine *A* just puts the bits on the wire, and machine *B* just takes them off. Unfortunately, communication circuits make errors occasionally. Furthermore, they have only a finite data rate, and there is a nonzero propagation delay between the time a bit is sent and the time it is received. These limitations have important implications for the efficiency of the data transfer. The protocols used for communications must take all these factors into consideration. These protocols are the subject of this chapter.

After an introduction to the key design issues present in the data link layer, we will start our study of its protocols by looking at the nature of errors, their causes, and how they can be detected and corrected. Then we will study a series of increasingly complex protocols, each one solving more and more of the problems present in this layer. Finally, we will conclude with an examination of protocol modeling and correctness and give some examples of data link protocols.

3.1. DATA LINK LAYER DESIGN ISSUES

The data link layer has a number of specific functions to carry out. These functions include providing a well-defined service interface to the network layer, determining how the bits of the physical layer are grouped into frames, dealing with transmission errors, and regulating the flow of frames so that slow receivers are not swamped by fast senders. In the following sections we will examine each of these issues in turn.

3.1.1. Services Provided to the Network Layer

The function of the data link layer is to provide services to the network layer. The principal service is transferring data from the network layer on the source machine to the network layer on the destination machine. On the source machine there is an entity, call it a process, in the network layer that hands some bits to the data link layer for transmission to the destination. The job of the data link layer is to transmit the bits to the destination machine, so they can be handed over to the network layer there, as shown in Fig. 3-1(a). The actual transmission follows the path of Fig. 3-1(b), but it is easier to think in terms of two data link layer processes communicating using a data link protocol. For this reason, we will implicitly use the model of Fig. 3-1(a) throughout this chapter.

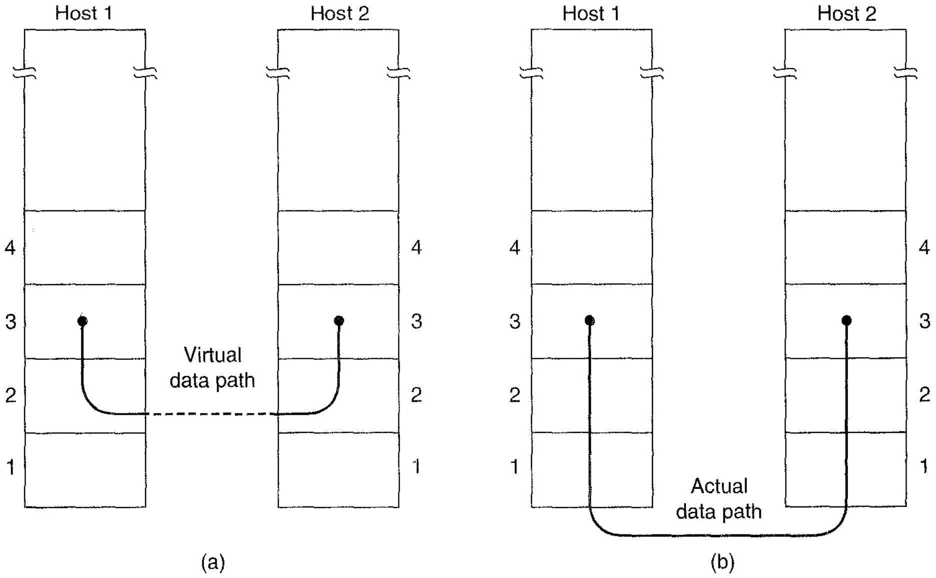


Fig. 3-1. (a) Virtual communication. (b) Actual communication.

The data link layer can be designed to offer various services. The actual services offered can vary from system to system. Three reasonable possibilities that are commonly provided are

1. Unacknowledged connectionless service.
2. Acknowledged connectionless service.
3. Acknowledged connection-oriented service.

Let us consider each of these in turn.

Unacknowledged connectionless service consists of having the source machine send independent frames to the destination machine without having the destination machine acknowledge them. No connection is established beforehand or released afterward. If a frame is lost due to noise on the line, no attempt is made to recover it in the data link layer. This class of service is appropriate when the error rate is very low so recovery is left to higher layers. It is also appropriate for real-time traffic, such as speech, in which late data are worse than bad data. Most LANs use unacknowledged connectionless service in the data link layer.

The next step up in terms of reliability is acknowledged connectionless service. When this service is offered, there are still no connections used, but each frame sent is individually acknowledged. In this way, the sender knows whether or not a frame has arrived safely. If it has not arrived within a specified time interval, it can be sent again. This service is useful over unreliable channels, such as wireless systems.

It is perhaps worth emphasizing that providing acknowledgements in the data link layer is just an optimization, never a requirement. The transport layer can always send a message and wait for it to be acknowledged. If the acknowledgement is not forthcoming before the timer goes off, the sender can just send the entire message again. The trouble with this strategy is that if the average message is broken up into, say, 10 frames, and 20 percent of all frames are lost, it may take a very long time for the message to get through. If individual frames are acknowledged and retransmitted, entire messages get through much faster. On reliable channels, such as fiber, the overhead of a heavyweight data link protocol may be unnecessary, but on wireless channels it is well worth the cost due to their inherent unreliability.

Getting back to our services, the most sophisticated service the data link layer can provide to the network layer is connection-oriented service. With this service, the source and destination machines establish a connection before any data are transferred. Each frame sent over the connection is numbered, and the data link layer guarantees that each frame sent is indeed received. Furthermore, it guarantees that each frame is received exactly once and that all frames are received in the right order. With connectionless service, in contrast, it is conceivable that a lost acknowledgement causes a frame to be sent several times and thus received several times. Connection-oriented service, in contrast, provides the network layer processes with the equivalent of a reliable bit stream.

When connection-oriented service is used, transfers have three distinct phases. In the first phase the connection is established by having both sides initialize variables and counters needed to keep track of which frames have been received and which ones have not. In the second phase, one or more frames are actually transmitted. In the third and final phase, the connection is released, freeing up the variables, buffers, and other resources used to maintain the connection.

Consider a typical example: a WAN subnet consisting of routers connected by point-to-point leased telephone lines. When a frame arrives at a router, the hardware verifies the checksum and passes the frame to the data link layer software (which might be embedded in a chip on the network adaptor board). The data link layer software checks to see if this is the frame expected, and if so, gives the packet contained in the payload field to the routing software. The routing software chooses the appropriate outgoing line and passes the packet back down to the data link layer software, which then transmits it. The flow over two routers is shown in Fig. 3-2.

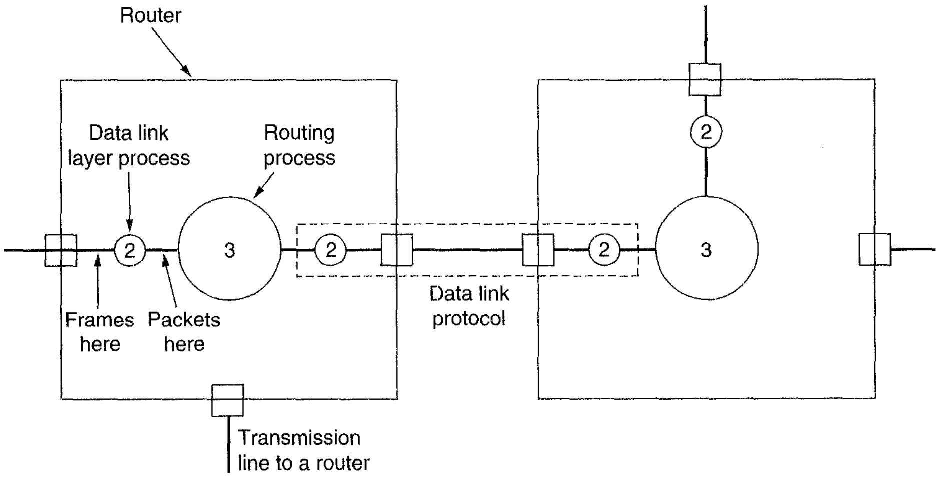


Fig. 3-2. Placement of the data link protocol.

The routing code frequently wants the job done right, that is, reliable, sequenced connections on each of the point-to-point lines. It does not want to be bothered too often with packets that got lost on the way. It is up to the data link protocol, shown in the dotted rectangle, to make unreliable communication lines look perfect, or at least, fairly good. This property is especially important for wireless links, which are inherently very unreliable. As an aside, although we have shown multiple copies of the data link layer software in each router, in fact, one copy handles all the lines, with different tables and data structures for each one.

Although this chapter is explicitly about the data link layer and the data link protocols, many of the principles we will study here, such as error control and flow control, are also found in transport and other protocols as well.

3.1.2. Framing

In order to provide service to the network layer, the data link layer must use the service provided to it by the physical layer. What the physical layer does is accept a raw bit stream and attempt to deliver it to the destination. This bit stream is not guaranteed to be error free. The number of bits received may be less than, equal to, or more than the number of bits transmitted, and they may have different values. It is up to the data link layer to detect, and if necessary, correct errors.

The usual approach is for the data link layer to break the bit stream up into discrete frames and compute the checksum for each frame. (Checksum algorithms will be discussed later in this chapter.) When a frame arrives at the destination, the checksum is recomputed. If the newly computed checksum is different from the one contained in the frame, the data link layer knows that an error has occurred and takes steps to deal with it (e.g., discarding the bad frame and sending back an error report).

Breaking the bit stream up into frames is more difficult than it at first appears. One way to achieve this framing is to insert time gaps between frames, much like the spaces between words in ordinary text. However, networks rarely make any guarantees about timing, so it is possible these gaps might be squeezed out, or other gaps might be inserted during transmission.

Since it is too risky to count on timing to mark the start and end of each frame, other methods have been devised. In this section we will look at four methods:

1. Character count.
2. Starting and ending characters, with character stuffing.
3. Starting and ending flags, with bit stuffing.
4. Physical layer coding violations.

The first framing method uses a field in the header to specify the number of characters in the frame. When the data link layer at the destination sees the character count, it knows how many characters follow, and hence where the end of the frame is. This technique is shown in Fig. 3-3(a) for four frames of sizes 5, 5, 8, and 9 characters respectively.

The trouble with this algorithm is that the count can be garbled by a transmission error. For example, if the character count of 5 in the second frame of Fig. 3-3(b) becomes a 7, the destination will get out of synchronization and will be unable to locate the start of the next frame. Even if the checksum is incorrect so the destination knows that the frame is bad, it still has no way of telling where the

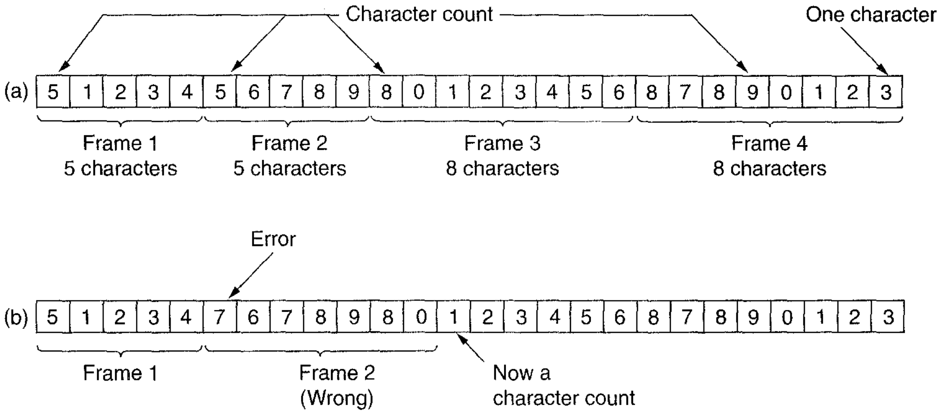


Fig. 3-3. A character stream. (a) Without errors. (b) With one error.

next frame starts. Sending a frame back to the source asking for a retransmission does not help either, since the destination does not know how many characters to skip over to get to the start of the retransmission. For this reason, the character count method is rarely used anymore.

The second framing method gets around the problem of resynchronization after an error by having each frame start with the ASCII character sequence DLE STX and end with the sequence DLE ETX. (DLE is Data Link Escape, STX is Start of TeXt, and ETX is End of TeXt.) In this way, if the destination ever loses track of the frame boundaries, all it has to do is look for DLE STX or DLE ETX characters to figure out where it is.

A serious problem occurs with this method when binary data, such as object programs or floating-point numbers, are being transmitted. It may easily happen that the characters for DLE STX or DLE ETX occur in the data, which will interfere with the framing. One way to solve this problem is to have the sender's data link layer insert an ASCII DLE character just before each "accidental" DLE character in the data. The data link layer on the receiving end removes the DLE before the data are given to the network layer. This technique is called **character stuffing**. Thus a framing DLE STX or DLE ETX can be distinguished from one in the data by the absence or presence of a single DLE. DLEs in the data are always doubled. Figure 3-4 gives an example data stream before stuffing, after stuffing, and after destuffing.

A major disadvantage of using this framing method is that it is closely tied to 8-bit characters in general and the ASCII character code in particular. As networks developed, the disadvantages of embedding the character code in the framing mechanism became more and more obvious so a new technique had to be developed to allow arbitrary sized characters.

The new technique allows data frames to contain an arbitrary number of bits

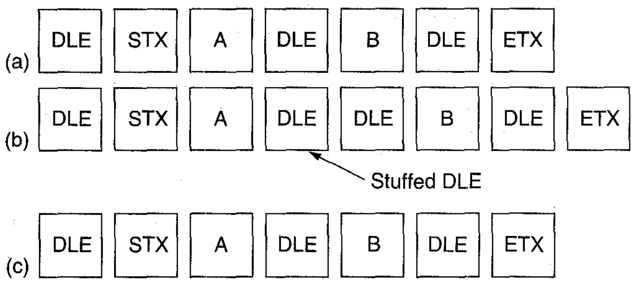


Fig. 3-4. (a) Data sent by the network layer. (b) Data after being character stuffed by the data link layer. (c) Data passed to the network layer on the receiving side.

and allows character codes with an arbitrary number of bits per character. It works like this. Each frame begins and ends with a special bit pattern, 01111110, called a **flag** byte. Whenever the sender's data link layer encounters five consecutive ones in the data, it automatically stuffs a 0 bit into the outgoing bit stream. This **bit stuffing** is analogous to character stuffing, in which a DLE is stuffed into the outgoing character stream before DLE in the data.

When the receiver sees five consecutive incoming 1 bits, followed by a 0 bit, it automatically destuffs (i.e., deletes) the 0 bit. Just as character stuffing is completely transparent to the network layer in both computers, so is bit stuffing. If the user data contain the flag pattern, 01111110, this flag is transmitted as 011111010 but stored in the receiver's memory as 01111110. Figure 3-5 gives an example of bit stuffing.

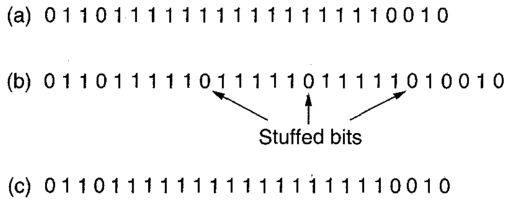


Fig. 3-5. Bit stuffing. (a) The original data. (b) The data as they appear on the line. (c) The data as they are stored in the receiver's memory after destuffing.

With bit stuffing, the boundary between two frames can be unambiguously recognized by the flag pattern. Thus if the receiver loses track of where it is, all it has to do is scan the input for flag sequences, since they can only occur at frame boundaries and never within the data.

The last method of framing is only applicable to networks in which the encoding on the physical medium contains some redundancy. For example, some LANs encode 1 bit of data by using 2 physical bits. Normally, a 1 bit is a high-low pair and a 0 bit is a low-high pair. The combinations high-high and low-low are not used for data. The scheme means that every data bit has a transition in the middle, making it easy for the receiver to locate the bit boundaries. This use of invalid physical codes is part of the 802 LAN standard, which we will study in Chap. 4.

As a final note on framing, many data link protocols use a combination of a character count with one of the other methods for extra safety. When a frame arrives, the count field is used to locate the end of the frame. Only if the appropriate delimiter is present at that position and the checksum is correct, is the frame accepted as valid. Otherwise, the input stream is scanned for the next delimiter.

3.1.3. Error Control

Having solved the problem of marking the start and end of each frame, we come to the next problem: how to make sure all frames are eventually delivered to the network layer at the destination, and in the proper order. Suppose that the sender just kept outputting frames without regard to whether they were arriving properly. This might be fine for unacknowledged connectionless service but would most certainly not be fine for reliable, connection-oriented service.

The usual way to ensure reliable delivery is to provide the sender with some feedback about what is happening at the other end of the line. Typically the protocol calls for the receiver to send back special control frames bearing positive or negative acknowledgements about the incoming frames. If the sender receives a positive acknowledgement about a frame, it knows the frame has arrived safely. On the other hand, a negative acknowledgement means that something has gone wrong, and the frame must be transmitted again.

An additional complication comes from the possibility that hardware troubles may cause a frame to vanish completely (e.g., in a noise burst). In this case, the receiver will not react at all, since it has no reason to react. It should be clear that a protocol in which the sender transmitted a frame and then waited for an acknowledgement, positive or negative, would hang forever if a frame were ever completely lost due to malfunctioning hardware.

This possibility is dealt with by introducing timers into the data link layer. When the sender transmits a frame, it generally also starts a timer. The timer is set to go off after an interval long enough for the frame to reach the destination, be processed there, and have the acknowledgement propagate back to the sender. Normally, the frame will be correctly received and the acknowledgement will get back before the timer runs out, in which case it will be canceled.

However, if either the frame or the acknowledgement is lost, the timer will go off, alerting the sender to a potential problem. The obvious solution is to just transmit the frame again. However, when frames may be transmitted multiple times there is a danger that the receiver will accept the same frame two or more times, and pass it to the network layer more than once. To prevent this from happening, it is generally necessary to assign sequence numbers to outgoing frames, so that the receiver can distinguish retransmissions from originals.

The whole issue of managing the timers and sequence numbers so as to ensure that each frame is ultimately passed to the network layer at the destination exactly once, no more and no less, is an important part of the data link layer's duties. Later in this chapter, we will study in detail how this management is done by looking at a series of increasingly sophisticated examples.

3.1.4. Flow Control

Another important design issue that occurs in the data link layer (and higher layers as well) is what to do with a sender that systematically wants to transmit frames faster than the receiver can accept them. This situation can easily occur when the sender is running on a fast (or lightly loaded) computer and the receiver is running on a slow (or heavily loaded) machine. The sender keeps pumping the frames out at a high rate until the receiver is completely swamped. Even if the transmission is error free, at a certain point the receiver will simply not be able to handle the frames as they arrive and will start to lose some. Clearly, something has to be done to prevent this situation.

The usual solution is to introduce **flow control** to throttle the sender into sending no faster than the receiver can handle the traffic. This throttling generally requires some kind of a feedback mechanism, so the sender can be made aware of whether or not the receiver is able to keep up.

Various flow control schemes are known, but most of them use the same basic principle. The protocol contains well-defined rules about when a sender may transmit the next frame. These rules often prohibit frames from being sent until the receiver has granted permission, either implicitly or explicitly. For example, when a connection is set up, the receiver might say: "You may send me *n* frames now, but after they have been sent, do not send any more until I have told you to continue." In this chapter, we will study various flow control mechanisms based on this principle. In subsequent chapters, we will study other mechanisms.

3.2. ERROR DETECTION AND CORRECTION

As we saw in Chap. 2, the telephone system has three parts: the switches, the interoffice trunks, and the local loops. The first two are now almost entirely digital in the United States and some other countries. The local loops are still analog twisted copper pairs everywhere and will continue to be so for decades due to the enormous expense of replacing them. While errors are rare on the digital part, they are still common on the local loops. Furthermore, wireless communication is becoming more common, and the error rates here are orders of magnitude worse than on the interoffice fiber trunks. The conclusion is: transmission errors are going to be a fact of life for many years to come.

As a result of the physical processes that generate them, errors on some media (e.g., radio) tend to come in bursts rather than singly. Having the errors come in bursts has both advantages and disadvantages over isolated single-bit errors. On the advantage side, computer data are always sent in blocks of bits. Suppose that the block size is 1000 bits, and the error rate is 0.001 per bit. If errors were independent, most blocks would contain an error. If the errors came in bursts of 100 however, only one or two blocks in 100 would be affected, on the average. The disadvantage of burst errors is that they are much harder to detect and correct than are isolated errors.

3.2.1. Error-Correcting Codes

Network designers have developed two basic strategies for dealing with errors. One way is to include enough redundant information along with each block of data sent to enable the receiver to deduce what the transmitted character must have been. The other way is to include only enough redundancy to allow the receiver to deduce that an error occurred, but not which error, and have it request a retransmission. The former strategy uses **error-correcting codes** and the latter uses **error-detecting codes**.

To understand how errors can be handled, it is necessary to look closely at what an error really is. Normally, a frame consists of m data (i.e., message) bits and r redundant, or check bits. Let the total length be n (i.e., $n = m + r$). An n-bit unit containing data and checkbits is often referred to as an n-bit **codeword**.

Given any two codewords, say, 10001001 and 10110001, it is possible to determine how many corresponding bits differ. In this case, 3 bits differ. To determine how many bits differ, just EXCLUSIVE OR the two codewords, and count the number of 1 bits in the result. The number of bit positions in which two codewords differ is called the **Hamming distance** (Hamming, 1950). Its significance is that if two codewords are a Hamming distance d apart, it will require d single-bit errors to convert one into the other.

In most data transmission applications, all 2^m possible data messages are legal, but due to the way the check bits are computed, not all of the 2^n possible codewords are used. Given the algorithm for computing the check bits, it is possible to construct a complete list of the legal codewords, and from this list find the two codewords whose Hamming distance is minimum. This distance is the Hamming distance of the complete code.

The error-detecting and error-correcting properties of a code depend on its Hamming distance. To detect d errors, you need a distance $d + 1$ code because with such a code there is no way that d single-bit errors can change a valid codeword into another valid codeword. When the receiver sees an invalid codeword, it can tell that a transmission error has occurred. Similarly, to correct d errors, you need a distance $2d + 1$ code because that way the legal codewords are so far apart that even with d changes, the original codeword is still closer than any other codeword, so it can be uniquely determined.

As a simple example of an error-detecting code, consider a code in which a single **parity bit** is appended to the data. The parity bit is chosen so that the number of 1 bits in the codeword is even (or odd). For example, when 10110101 is sent in even parity by adding a bit at the end, it becomes 101101011, whereas 10110001 becomes 101100010 with even parity. A code with a single parity bit has a distance 2, since any single-bit error produces a codeword with the wrong parity. It can be used to detect single errors.

As a simple example of an error-correcting code, consider a code with only four valid codewords:

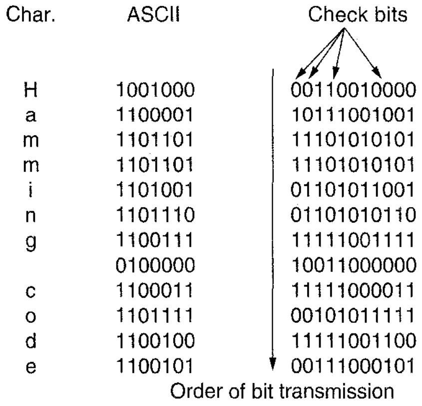
0000000000, 0000011111, 1111100000, and 1111111111

This code has a distance 5, which means that it can correct double errors. If the codeword 0000000111 arrives, the receiver knows that the original must have been 0000011111. If, however, a triple error changes 0000000000 into 0000000111, the error will not be corrected properly.

Imagine that we want to design a code with m message bits and r check bits that will allow all single errors to be corrected. Each of the 2^m legal messages has n illegal codewords at a distance 1 from it. These are formed by systematically inverting each of the n bits in the n-bit codeword formed from it. Thus each of the 2^m legal messages requires $n + 1$ bit patterns dedicated to it. Since the total number of bit patterns is 2^n, we must have $(n + 1)2^m \leq 2^n$. Using $n = m + r$, this requirement becomes $(m + r + 1) \leq 2^r$. Given m, this puts a lower limit on the number of check bits needed to correct single errors.

This theoretical lower limit can, in fact, be achieved using a method due to Hamming (1950). The bits of the codeword are numbered consecutively, starting with bit 1 at the left end. The bits that are powers of 2 (1, 2, 4, 8, 16, etc.) are check bits. The rest (3, 5, 6, 7, 9, etc.) are filled up with the m data bits. Each check bit forces the parity of some collection of bits, including itself, to be even (or odd). A bit may be included in several parity computations. To see which check bits the data bit in position k contributes to, rewrite k as a sum of powers of 2. For example, $11 = 1 + 2 + 8$ and $29 = 1 + 4 + 8 + 16$. A bit is checked by just those check bits occurring in its expansion (e.g., bit 11 is checked by bits 1, 2, and 8).

When a codeword arrives, the receiver initializes a counter to zero. It then examines each check bit, k ($k = 1, 2, 4, 8, \ldots$) to see if it has the correct parity. If not, it adds k to the counter. If the counter is zero after all the check bits have been examined (i.e., if they were all correct), the codeword is accepted as valid. If the counter is nonzero, it contains the number of the incorrect bit. For example, if check bits 1, 2, and 8 are in error, the inverted bit is 11, because it is the only



Char.	ASCII	Check bits
H	1001000	00110010000
a	1100001	10111001001
m	1101101	11101010101
m	1101101	11101010101
i	1101001	01101011001
n	1101110	01101010110
g	1100111	11111001111
	0100000	10011000000
c	1100011	11111000011
o	1101111	00101011111
d	1100100	11111001100
e	1100101	00111000101

Fig. 3-6. Use of a Hamming code to correct burst errors.

one checked by bits 1, 2, and 8. Figure 3-6 shows some 7-bit ASCII characters encoded as 11-bit codewords using a Hamming code. Remember that the data are found in bit positions 3, 5, 6, 7, 9, 10, and 11.

Hamming codes can only correct single errors. However, there is a trick that can be used to permit Hamming codes to correct burst errors. A sequence of k consecutive codewords are arranged as a matrix, one codeword per row. Normally, the data would be transmitted one codeword at a time, from left to right. To correct burst errors, the data should be transmitted one column at a time, starting with the leftmost column. When all k bits have been sent, the second column is sent, and so on. When the frame arrives at the receiver, the matrix is reconstructed, one column at a time. If a burst error of length k occurs, at most 1 bit in each of the k codewords will have been affected, but the Hamming code can correct one error per codeword, so the entire block can be restored. This method uses kr check bits to make blocks of km data bits immune to a single burst error of length k or less.

3.2.2. Error-Detecting Codes

Error-correcting codes are sometimes used for data transmission, for example, when the channel is simplex, so retransmissions cannot be requested, but most often error detection followed by retransmission is preferred because it is more efficient. As a simple example, consider a channel on which errors are isolated and the error rate is 10^{-6} per bit. Let the block size be 1000 bits. To provide error correction for 1000-bit blocks, 10 check bits are needed; a megabit of data would require 10,000 check bits. To merely detect a block with a single 1-bit error, one parity bit per block will suffice. Once every 1000 blocks an extra block (1001 bits) will have to be transmitted. The total overhead for the error detection +

retransmission method is only 2001 bits per megabit of data, versus 10,000 bits for a Hamming code.

If a single parity bit is added to a block and the block is badly garbled by a long burst error, the probability that the error will be detected is only 0.5, which is hardly acceptable. The odds can be improved considerably by regarding each block to be sent as a rectangular matrix n bits wide and k bits high. A parity bit is computed separately for each column and affixed to the matrix as the last row. The matrix is then transmitted one row at a time. When the block arrives, the receiver checks all the parity bits. If any one of them is wrong, it requests a retransmission of the block.

This method can detect a single burst of length n, since only 1 bit per column will be changed. A burst of length $n + 1$ will pass undetected, however, if the first bit is inverted, the last bit is inverted, and all the other bits are correct. (A burst error does not imply that all the bits are wrong; it just implies that at least the first and last are wrong.) If the block is badly garbled by a long burst or by multiple shorter bursts, the probability that any of the n columns will have the correct parity, by accident, is 0.5, so the probability of a bad block being accepted when it should not be is 2^{-n}.

Although the above scheme may sometimes be adequate, in practice, another method is in widespread use: the **polynomial code** (also known as a **cyclic redundancy code** or CRC code). Polynomial codes are based upon treating bit strings as representations of polynomials with coefficients of 0 and 1 only. A k-bit frame is regarded as the coefficient list for a polynomial with k terms, ranging from x^{k-1} to x^0. Such a polynomial is said to be of degree $k - 1$. The high-order (leftmost) bit is the coefficient of x^{k-1}; the next bit is the coefficient of x^{k-2}, and so on. For example, 110001 has 6 bits and thus represents a six-term polynomial with coefficients 1, 1, 0, 0, 0, and 1: $x^5 + x^4 + x^0$.

Polynomial arithmetic is done modulo 2, according to the rules of algebraic field theory. There are no carries for addition or borrows for subtraction. Both addition and subtraction are identical to EXCLUSIVE OR. For example:

$$\begin{array}{rrrr} 10011011 & 00110011 & 11110000 & 01010101 \\ +\ 11001010 & +\ 11001101 & -\ 10100110 & -\ 10101111 \\ \hline 01010001 & 11111110 & 01010110 & 11111010 \end{array}$$

Long division is carried out the same way as it is in binary except that the subtraction is done modulo 2, as above. A divisor is said "to go into" a dividend if the dividend has as many bits as the divisor.

When the polynomial code method is employed, the sender and receiver must agree upon a **generator polynomial**, $G(x)$, in advance. Both the high- and low-order bits of the generator must be 1. To compute the **checksum** for some frame with m bits, corresponding to the polynomial $M(x)$, the frame must be longer than the generator polynomial. The idea is to append a checksum to the end of the frame in such a way that the polynomial represented by the checksummed frame is divisible by $G(x)$. When the receiver gets the checksummed frame, it tries dividing it by $G(x)$. If there is a remainder, there has been a transmission error.

The algorithm for computing the checksum is as follows:

1. Let r be the degree of $G(x)$. Append r zero bits to the low-order end of the frame, so it now contains $m + r$ bits and corresponds to the polynomial $x^r M(x)$.
2. Divide the bit string corresponding to $G(x)$ into the bit string corresponding to $x^r M(x)$ using modulo 2 division.
3. Subtract the remainder (which is always r or fewer bits) from the bit string corresponding to $x^r M(x)$ using modulo 2 subtraction. The result is the checksummed frame to be transmitted. Call its polynomial $T(x)$.

Figure 3-7 illustrates the calculation for a frame 1101011011 and $G(x) = x^4 + x + 1$.

It should be clear that $T(x)$ is divisible (modulo 2) by $G(x)$. In any division problem, if you diminish the dividend by the remainder, what is left over is divisible by the divisor. For example, in base 10, if you divide 210,278 by 10,941, the remainder is 2399. By subtracting off 2399 from 210,278, what is left over (207,879) is divisible by 10,941.

Now let us analyze the power of this method. What kinds of errors will be detected? Imagine that a transmission error occurs, so that instead of the bit string for $T(x)$ arriving, $T(x) + E(x)$ arrives. Each 1 bit in $E(x)$ corresponds to a bit that has been inverted. If there are k 1 bits in $E(x)$, k single-bit errors have occurred. A single burst error is characterized by an initial 1, a mixture of 0s and 1s, and a final 1, with all other bits being 0.

Upon receiving the checksummed frame, the receiver divides it by $G(x)$; that is, it computes $[T(x) + E(x)]/G(x)$. $T(x)/G(x)$ is 0, so the result of the computation is simply $E(x)/G(x)$. Those errors that happen to correspond to polynomials containing $G(x)$ as a factor will slip by; all other errors will be caught.

If there has been a single-bit error, $E(x) = x^i$, where i determines which bit is in error. If $G(x)$ contains two or more terms, it will never divide $E(x)$, so all single-bit errors will be detected.

If there have been two isolated single-bit errors, $E(x) = x^i + x^j$, where $i > j$. Alternatively, this can be written as $E(x) = x^j(x^{i-j} + 1)$. If we assume that $G(x)$ is not divisible by x, a sufficient condition for all double errors to be detected is that $G(x)$ does not divide $x^k + 1$ for any k up to the maximum value of $i - j$ (i.e., up to the maximum frame length). Simple, low-degree polynomials that give protection to long frames are known. For example, $x^{15} + x^{14} + 1$ will not divide $x^k + 1$ for any value of k below 32,768.

If there are an odd number of bits in error, $E(X)$ contains an odd number of terms (e.g., $x^5 + x^2 + 1$, but not $x^2 + 1$). Interestingly enough, there is no

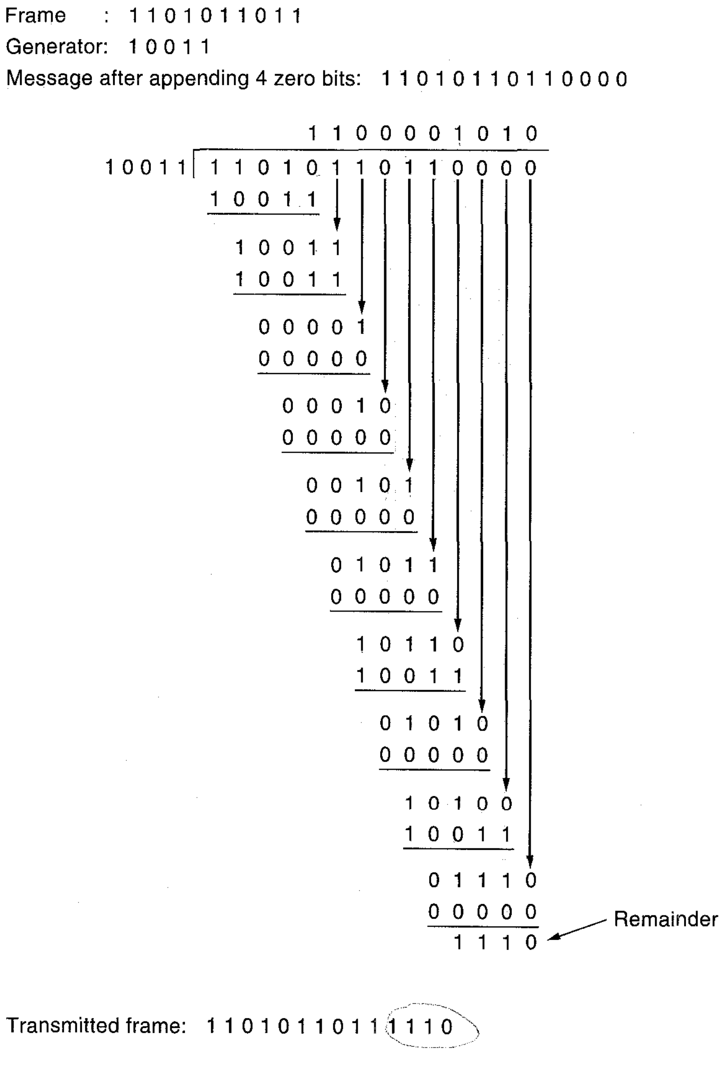


Fig. 3-7. Calculation of the polynomial code checksum.

polynomial with an odd number of terms that has $x + 1$ as a factor in the modulo 2 system. By making $x + 1$ a factor of $G(x)$, we can catch all errors consisting of an odd number of inverted bits.

To see that no polynomial with an odd number of terms is divisible by $x + 1$, assume that $E(x)$ has an odd number of terms and is divisible by $x + 1$. Factor $E(x)$ into $(x + 1)\ Q(x)$. Now evaluate $E(1) = (1 + 1)Q(1)$. Since $1 + 1 = 0$ (modulo 2), $E(1)$ must be zero. If $E(x)$ has an odd number of terms, substituting 1 for x everywhere will always yield 1 as result. Thus no polynomial with an odd number of terms is divisible by $x + 1$.

Finally, and most important, a polynomial code with r check bits will detect all burst errors of length $\leq r$. A burst error of length k can be represented by $x^i(x^{k-1} + \ldots + 1)$, where i determines how far from the right-hand end of the received frame the burst is located. If $G(x)$ contains an x^0 term, it will not have x^i as a factor, so if the degree of the parenthesized expression is less than the degree of $G(x)$, the remainder can never be zero.

If the burst length is $r + 1$, the remainder of the division by $G(x)$ will be zero if and only if the burst is identical to $G(x)$. By definition of a burst, the first and last bits must be 1, so whether it matches depends on the $r - 1$ intermediate bits. If all combinations are regarded as equally likely, the probability of such an incorrect frame being accepted as valid is $\frac{1}{2}^{r-1}$.

It can also be shown that when an error burst longer than $r + 1$ bits occurs, or several shorter bursts occur, the probability of a bad frame getting through unnoticed is $\frac{1}{2}^r$ assuming that all bit patterns are equally likely.

Three polynomials have become international standards:

CRC–12 $= x^{12} + x^{11} + x^3 + x^2 + x^1 + 1$
CRC–16 $= x^{16} + x^{15} + x^2 + 1$ 0x3EEF
CRC–CCITT $= x^{16} + x^{12} + x^5 + 1$

All three contain $x + 1$ as a prime factor. CRC-12 is used when the character length is 6 bits. The other two are used for 8-bit characters. A 16-bit checksum, such as CRC-16 or CRC-CCITT, catches all single and double errors, all errors with an odd number of bits, all burst errors of length 16 or less, 99.997 percent of 17-bit error bursts, and 99.998 percent of 18-bit and longer bursts.

Although the calculation required to compute the checksum may seem complicated, Peterson and Brown (1961) have shown that a simple shift register circuit can be constructed to compute and verify the checksums in hardware. In practice, this hardware is nearly always used.

For decades, it has been assumed that frames to be checksummed contain random bits. All analyses of checksum algorithms have been made under this assumption. More recently inspection of real data has shown this assumption to be quite wrong. As a consequence, under some circumstances, undetected errors are much more common than had been previously thought (Partridge et al., 1995).

3.3. ELEMENTARY DATA LINK PROTOCOLS

To introduce the subject of protocols, we will begin by looking at three protocols of increasing complexity. For interested readers, a simulator for these and subsequent protocols is available via the WWW (see the preface). Before we look at the protocols, it is useful to make explicit some of the assumptions underlying the model of communication. To start with, we are assuming that in the physical layer, data link layer, and network layer are independent processes that communicate by passing messages back and forth. In some cases, the physical and data link layer processes will be running on a processor inside a special network I/O chip and the network layer on the main CPU, but other implementations are also possible (e.g., three processes inside a single I/O chip; the physical and data link layers as procedures called by the network layer process, and so on). In any event, treating the three layers as separate processes makes the discussion conceptually cleaner and also serves to emphasize the independence of the layers.

Another key assumption is that machine *A* wants to send a long stream of data to machine *B* using a reliable, connection-oriented service. Later, we will consider the case where *B* also wants to send data to *A* simultaneously. *A* is assumed to have an infinite supply of data ready to send and never has to wait for data to be produced. When *A*'s data link layer asks for data, the network layer is always able to comply immediately. (This restriction, too, will be dropped later.)

As far as the data link layer is concerned, the packet passed across the interface to it from the network layer is pure data, every bit of which is to be delivered to the destination's network layer. The fact that the destination's network layer may interpret part of the packet as a header is of no concern to the data link layer.

When the data link layer accepts a packet, it encapsulates the packet in a frame by adding a data link header and trailer to it (see Fig. 1-11). Thus a frame consists of an embedded packet and some control (header) information. The frame is then transmitted to the other data link layer. We will assume that there exist suitable library procedures *to_physical_layer* to send a frame and *from_physical_layer* to receive a frame. The transmitting hardware computes and appends the checksum, so that the data link layer software need not worry about it. The polynomial algorithm discussed earlier in this chapter might be used, for example.

Initially, the receiver has nothing to do. It just sits around waiting for something to happen. In the example protocols of this chapter we indicate that the data link layer is waiting for something to happen by the procedure call *wait_for_event(&event)*. This procedure only returns when something has happened (e.g., a frame has arrived). Upon return, the variable *event* tells what happened. The set of possible events differs for the various protocols to be described and will be defined separately for each protocol. Note that in a more realistic situation, the data link layer will not sit in a tight loop waiting for an event, as we have suggested, but will receive an interrupt, which will cause it to stop whatever it was doing and go handle the incoming frame. Nevertheless, for simplicity we will ignore all the details of parallel activity within the data link layer and assume that it is dedicated full time to handling just our one channel.

When a frame arrives at the receiver, the hardware computes the checksum. If the checksum is incorrect (i.e., there was a transmission error), the data link layer is so informed (*event* = *cksum_err*). If the inbound frame arrived undamaged, the data link layer is also informed (*event* = *frame_arrival*), so it can acquire the frame for inspection using *from_physical_layer*. As soon as the receiving data link layer has acquired an undamaged frame, it checks the control information in the header, and if everything is all right, the packet portion is passed to the network layer. Under no circumstances is a frame header ever given to a network layer.

There is a good reason why the network layer must never be given any part of the frame header: to keep the network and data link protocols completely separate. As long as the network layer knows nothing at all about the data link protocol or the frame format, these things can be changed without requiring changes to the network layer's software. Providing a rigid interface between network layer and data link layer greatly simplifies the software design, because communication protocols in different layers can evolve independently.

Figure 3-8 shows some declarations (in C) common to many of the protocols to be discussed later. Five data structures are defined there: *boolean*, *seq_nr*, *packet*, *frame_kind*, and *frame*. A *boolean* is an enumerated type and can take on the values *true* and *false*. A *seq_nr* is a small integer used to number the frames, so we can tell them apart. These sequence numbers run from 0 up to and including *MAX_SEQ*, which is defined in each protocol needing it. A *packet* is the unit of information exchanged between the network layer and the data link layer on the same machine, or between network layer peers. In our model it always contains *MAX_PKT* bytes, but more realistically it would be of variable length.

A *frame* is composed of four fields: *kind*, *seq*, *ack*, and *info*, the first three of which contain control information, and the last of which may contain actual data to be transferred. These control fields are collectively called the **frame header**. The *kind* field tells whether or not there are any data in the frame, because some of the protocols distinguish frames containing exclusively control information from those containing data as well. The *seq* and *ack* fields are used for sequence numbers and acknowledgements, respectively; their use will be described in more detail later. The *info* field of a data frame contains a single packet; the *info* field of a control frame is not used. A more realistic implementation would use a variable-length *info* field, omitting it altogether for control frames.

It is important to realize the relationship between a packet and a frame. The network layer builds a packet by taking a message from the transport layer and adding the network layer header to it. This packet is passed to the data link layer for inclusion in the *info* field of an outgoing frame. When the frame arrives at the destination, the data link layer extracts the packet from the frame and passes the packet to the network layer. In this manner, the network layer can act as though machines can exchange packets directly.

A number of procedures are also listed in Fig. 3-8. These are library routines whose details are implementation-dependent and whose inner workings will not concern us further here. The procedure *wait_for_event* sits in a tight loop waiting for something to happen, as mentioned earlier. The procedures *to_network_layer* and *from_network_layer* are used by the data link layer to pass packets to the network layer and accept packets from the network layer, respectively. Note that *from_physical_layer* and *to_physical_layer* are used for passing frames between the data link and physical layers, whereas the procedures *to_network_layer* and *from_network_layer* are used for passing packets between the data link layer and network layer. In other words, *to_network_layer* and *from_network_layer* deal with the interface between layers 2 and 3, whereas *from_physical_layer* and *to_physical_layer* deal with the interface between layers 1 and 2.

In most of the protocols we assume an unreliable channel that loses entire frames upon occasion. To be able to recover from such calamities, the sending data link layer must start an internal timer or clock whenever it sends a frame. If no reply has been received within a certain predetermined time interval, the clock times out and the data link layer receives an interrupt signal.

In our protocols this is handled by allowing the procedure *wait_for_event* to return *event* = *timeout*. The procedures *start_timer* and *stop_timer* are used to turn the timer on and off, respectively. Timeouts are possible only when the timer is running. It is explicitly permitted to call *start_timer* while the timer is running; such a call simply resets the clock to cause the next timeout after a full timer interval has elapsed (unless it is reset or turned off in the meanwhile).

The procedures *start_ack_timer* and *stop_ack_timer* are used to control an auxiliary timer used to generate acknowledgements under certain conditions.

The procedures *enable_network_layer* and *disable_network_layer* are used in the more sophisticated protocols, where we no longer assume that the network layer always has packets to send. When the data link layer enables the network layer, the network layer is then permitted to interrupt when it has a packet to be sent. We indicate this with *event* = *network_layer_ready*. When a network layer is disabled, it may not cause such events. By being careful about when it enables and disables its network layer, the data link layer can prevent the network layer from swamping it with packets for which it has no buffer space.

Frame sequence numbers are always in the range 0 to *MAX_SEQ* (inclusive), where *MAX_SEQ* is different for the different protocols. It is frequently necessary to advance a sequence number by 1 circularly (i.e., *MAX_SEQ* is followed by 0). The macro *inc* performs this incrementing. It has been defined as a macro because it is used in-line within the critical path. As we will see later in this book, the factor limiting network performance is often protocol processing, so defining simple operations like this as macros does not affect the readability of the code, but does improve performance. Also, since *MAX_SEQ* will have different values in different protocols, by making it a macro, it becomes possible to include all the protocols in the same binary without conflict. This ability is useful for the simulator.

The declarations of Fig. 3-8 are part of each of the protocols to follow. To save space and to provide a convenient reference, they have been extracted and

```
#define MAX_PKT 1024                    /* determines packet size in bytes */

typedef enum {false, true} boolean;     /* boolean type */
typedef unsigned int seq_nr;            /* sequence or ack numbers */
typedef struct {unsigned char data[MAX_PKT];} packet;/* packet definition */
typedef enum {data, ack, nak} frame_kind;   /* frame_kind definition */

typedef struct {                        /* frames are transported in this layer */
  frame_kind kind;                      /* what kind of a frame is it? */
  seq_nr seq;                           /* sequence number */
  seq_nr ack;                           /* acknowledgement number */
  packet info;                          /* the network layer packet */
} frame;

/* Wait for an event to happen; return its type in event. */
void wait_for_event(event_type *event);

/* Fetch a packet from the network layer for transmission on the channel. */
void from_network_layer(packet *p);

/* Deliver information from an inbound frame to the network layer. */
void to_network_layer(packet *p);

/* Go get an inbound frame from the physical layer and copy it to r. */
void from_physical_layer(frame *r);

/* Pass the frame to the physical layer for transmission. */
void to_physical_layer(frame *s);

/* Start the clock running and enable the timeout event. */
void start_timer(seq_nr k);

/* Stop the clock and disable the timeout event. */
void stop_timer(seq_nr k);

/* Start an auxiliary timer and enable the ack_timeout event. */
void start_ack_timer(void);

/* Stop the auxiliary timer and disable the ack_timeout event. */
void stop_ack_timer(void);

/* Allow the network layer to cause a network_layer_ready event. */
void enable_network_layer(void);

/* Forbid the network layer from causing a network_layer_ready event. */
void disable_network_layer(void);

/* Macro inc is expanded in-line: Increment k circularly. */
#define inc(k) if (k < MAX_SEQ) k = k + 1; else k = 0
```

Fig. 3-8. Some definitions needed in the protocols to follow. These definitions are located in the file *protocol.h*.

listed together, but conceptually they should be merged with the protocols themselves. In C, this merging is done by putting the definitions in a special header file, in this case *protocol.h*, and using the #include facility of the C preprocessor to include them in the protocol files.

3.3.1. An Unrestricted Simplex Protocol

As an initial example we will consider a protocol that is as simple as can be. Data are transmitted in one direction only. Both the transmitting and receiving network layers are always ready. Processing time can be ignored. Infinite buffer space is available. And best of all, the communication channel between the data link layers never damages or loses frames. This thoroughly unrealistic protocol, which we will nickname "utopia," is shown in Fig. 3-9.

The protocol consists of two distinct procedures, a sender and a receiver. The sender runs in the data link layer of the source machine, and the receiver runs in the data link layer of the destination machine. No sequence numbers or acknowledgements are used here, so *MAX_SEQ* is not needed. The only event type possible is *frame_arrival* (i.e., the arrival of an undamaged frame).

The sender is in an infinite while loop just pumping data out onto the line as fast as it can. The body of the loop consists of three actions: go fetch a packet from the (always obliging) network layer, construct an outbound frame using the variable *s*, and send the frame on its way. Only the *info* field of the frame is used by this protocol, because the other fields have to do with error and flow control, and there are no errors or flow control restrictions here.

The receiver is equally simple. Initially, it waits for something to happen, the only possibility being the arrival of an undamaged frame. Eventually, the frame arrives and the procedure *wait_for_event* returns, with *event* set to *frame_arrival* (which is ignored anyway). The call to *from_physical_layer* removes the newly arrived frame from the hardware buffer and puts it in the variable *r*. Finally, the data portion is passed on to the network layer and the data link layer settles back to wait for the next frame, effectively suspending itself until the frame arrives.

3.3.2. A Simplex Stop-and-Wait Protocol

Now we will drop the most unrealistic restriction used in protocol 1: the ability of the receiving network layer to process incoming data infinitely fast (or equivalently, the presence in the receiving data link layer of an infinite amount of buffer space in which to store all incoming frames while they are waiting their respective turns). The communication channel is still assumed to be error free however, and the data traffic is still simplex.

The main problem we have to deal with here is how to prevent the sender from flooding the receiver with data faster than the latter is able to process it. In essence, if the receiver requires a time Δt to execute *from_physical_layer* plus

```
/* Protocol 1 (utopia) provides for data transmission in one direction only, from
  sender to receiver. The communication channel is assumed to be error free,
  and the receiver is assumed to be able to process all the input infinitely fast.
  Consequently, the sender just sits in a loop pumping data out onto the line as
  fast as it can. */

typedef enum {frame_arrival} event_type;
#include "protocol.h"

void sender1(void)
{
 frame s;                          /* buffer for an outbound frame */
 packet buffer;                    /* buffer for an outbound packet */

 while (true) {
     from_network_layer(&buffer);  /* go get something to send */
     s.info = buffer;              /* copy it into s for transmission */
     to_physical_layer(&s);        /* send it on its way */
 }                                 /* Tomorrow, and tomorrow, and tomorrow,
                                     Creeps in this petty pace from day to day
                                     To the last syllable of recorded time
                                         - Macbeth, V, v */
}

void receiver1(void)
{
 frame r;
 event_type event;                 /* filled in by wait, but not used here */

 while (true) {
     wait_for_event(&event);       /* only possibility is frame_arrival */
     from_physical_layer(&r);      /* go get the inbound frame */
     to_network_layer(&r.info);    /* pass the data to the network layer */
 }
}
```

Fig. 3-9. An unrestricted simplex protocol.

to_network_layer, the sender must transmit at an average rate less than one frame per time Δt. Moreover, if we assume that there is no automatic buffering and queueing done within the receiver's hardware, the sender must never transmit a new frame until the old one has been fetched by *from_physical_layer*, lest the new one overwrite the old one.

In certain restricted circumstances (e.g., synchronous transmission and a receiving data link layer fully dedicated to processing the one input line), it might be possible for the sender to simply insert a delay into protocol 1 to slow it down sufficiently to keep from swamping the receiver. However, more usually, each data link layer will have several lines to attend to, and the time interval between a frame arriving and its being processed may vary considerably. If the network designers can calculate the worst-case behavior of the receiver, they can program the sender to transmit so slowly that even if every frame suffers the maximum delay, there will be no overruns. The trouble with this approach is that it is too conservative. It leads to a bandwidth utilization that is far below the optimum, unless the best and worst cases are almost the same (i.e., the variation in the data link layer's reaction time is small).

A more general solution to this dilemma is to have the receiver provide feedback to the sender. After having passed a packet to its network layer, the receiver sends a little dummy frame back to the sender which, in effect, gives the sender permission to transmit the next frame. After having sent a frame, the sender is required by the protocol to bide its time until the little dummy (i.e., acknowledgement) frame arrives.

Protocols in which the sender sends one frame and then waits for an acknowledgement before proceeding are called **stop-and-wait**. Figure 3-10 gives an example of a simplex stop-and-wait protocol.

As in protocol 1, the sender starts out by fetching a packet from the network layer, using it to construct a frame and sending it on its way. Only now, unlike in protocol 1, the sender must wait until an acknowledgement frame arrives before looping back and fetching the next packet from the network layer. The sending data link layer need not even inspect the incoming frame: there is only one possibility.

The only difference between *receiver1* and *receiver2* is that after delivering a packet to the network layer, *receiver2* sends an acknowledgement frame back to the sender before entering the wait loop again. Because only the arrival of the frame back at the sender is important, not its contents, the receiver need not put any particular information in it.

Although data traffic in this example is simplex, going only from the sender to the receiver, frames do travel in both directions. Consequently, the communication channel between the two data link layers needs to be capable of bidirectional information transfer. However, this protocol entails a strict alternation of flow: first the sender sends a frame, then the receiver sends a frame, then the sender sends another frame, then the receiver sends another one, and so on. A half-duplex physical channel would suffice here.

3.3.3. A Simplex Protocol for a Noisy Channel

Now let us consider the normal situation of a communication channel that makes errors. Frames may be either damaged or lost completely. However, we assume that if a frame is damaged in transit, the receiver hardware will detect this

```
/* Protocol 2 (stop-and-wait) also provides for a one-directional flow of data from
   sender to receiver. The communication channel is once again assumed to be error
   free, as in protocol 1. However, this time, the receiver has only a finite buffer
   capacity and a finite processing speed, so the protocol must explicitly prevent
   the sender from flooding the receiver with data faster than it can be handled. */

typedef enum {frame_arrival} event_type;
#include "protocol.h"

void sender2(void)
{
 frame s;                             /* buffer for an outbound frame */
 packet buffer;                       /* buffer for an outbound packet */
 event_type event;                    /* frame_arrival is the only possibility */

 while (true) {
     from_network_layer(&buffer);     /* go get something to send */
     s.info = buffer;                 /* copy it into s for transmission */
     to_physical_layer(&s);           /* bye bye little frame */
     wait_for_event(&event);          /* do not proceed until given the go ahead */
 }
}

void receiver2(void)
{
 frame r, s;                          /* buffers for frames */
 event_type event;                    /* frame_arrival is the only possibility */
 while (true) {
     wait_for_event(&event);          /* only possibility is frame_arrival */
     from_physical_layer(&r);         /* go get the inbound frame */
     to_network_layer(&r.info);       /* pass the data to the network layer */
     to_physical_layer(&s);           /* send a dummy frame to awaken sender */
 }
}
```

Fig. 3-10. A simplex stop-and-wait protocol.

when it computes the checksum. If the frame is damaged in such a way that the checksum is nevertheless correct, an exceedingly unlikely occurrence, this protocol (and all other protocols) can fail (i.e., deliver an incorrect packet to the network layer).

At first glance it might seem that a variation of protocol 2 would work: adding a timer. The sender could send a frame, but the receiver would only send an acknowledgement frame if the data were correctly received. If a damaged frame arrived at the receiver, it would be discarded. After a while the sender would time out and send the frame again. This process would be repeated until the frame finally arrived intact.

The above scheme has a fatal flaw in it. Think about the problem and try to discover what might go wrong before reading further.

To see what might go wrong, remember that it is the task of the data link layer processes to provide error free, transparent communication between network layers processes. The network layer on machine *A* gives a series of packets to its data link layer, which must ensure that an identical series of packets are delivered to the network layer on machine *B* by its data link layer. In particular, the network layer on *B* has no way of knowing that a packet has been lost or duplicated, so the data link layer must guarantee that no combination of transmission errors, no matter how unlikely, can cause a duplicate packet to be delivered to a network layer.

Consider the following scenario:

1. The network layer on *A* gives packet 1 to its data link layer. The packet is correctly received at *B* and passed to the network layer on *B*. *B* sends an acknowledgement frame back to *A*.
2. The acknowledgement frame gets lost completely. It just never arrives at all. Life would be a great deal simpler if the channel only mangled and lost data frames and not control frames, but sad to say, the channel is not very discriminating.
3. The data link layer on *A* eventually times out. Not having received an acknowledgement, it (incorrectly) assumes that its data frame was lost or damaged and sends the frame containing packet 1 again.
4. The duplicate frame also arrives at data link layer on *B* perfectly and is unwittingly passed to the network layer there. If *A* is sending a file to *B*, part of the file will be duplicated (i.e., the copy of the file made by *B* will be incorrect and the error will not have been detected). In other words, the protocol will fail.

Clearly, what is needed is some way for the receiver to be able to distinguish a frame that it is seeing for the first time from a retransmission. The obvious way to achieve this is to have the sender put a sequence number in the header of each frame it sends. Then the receiver can check the sequence number of each arriving frame to see if it is a new frame or a duplicate to be discarded.

Since a small frame header is desirable, the question arises: What is the minimum number of bits needed for the sequence number? The only ambiguity in this protocol is between a frame, m, and its direct successor, $m + 1$. If frame m is lost or damaged, the receiver will not acknowledge it, so the sender will keep trying to send it. Once it has been correctly received, the receiver will send an acknowledgement back to the sender. It is here that the potential trouble crops up. Depending upon whether the acknowledgement frame gets back to the sender correctly or not, the sender may try to send m or $m + 1$.

The event that triggers the sender to start sending $m + 2$ is the arrival of an acknowledgement for $m + 1$. But this implies that m has been correctly received, and furthermore that its acknowledgement has also been correctly received by the sender (otherwise, the sender would not have begun with $m + 1$, let alone $m + 2$). As a consequence, the only ambiguity is between a frame and its immediate predecessor or successor, not between the predecessor and successor themselves.

A 1-bit sequence number (0 or 1) is therefore sufficient. At each instant of time, the receiver expects a particular sequence number next. Any arriving frame containing the wrong sequence number is rejected as a duplicate. When a frame containing the correct sequence number arrives, it is accepted, passed to the network layer, and the expected sequence number is incremented modulo 2 (i.e., 0 becomes 1 and 1 becomes 0).

An example of this kind of protocol is shown in Fig. 3-11. Protocols in which the sender waits for a positive acknowledgement before advancing to the next data item are often called **PAR** (**Positive Acknowledgement with Retransmission**) or **ARQ** (**Automatic Repeat reQuest**). Like protocol 2, this one also transmits data only in one direction. Although it can handle lost frames (by timing out), it requires the timeout interval to be long enough to prevent premature timeouts. If the sender times out too early, while the acknowledgement is still on the way, it will send a duplicate.

When the previous acknowledgement finally does arrive, the sender will mistakenly think that the just-sent frame is the one being acknowledged and will not realize that there is potentially another acknowledgement frame somewhere "in the pipe." If the next frame sent is lost completely but the extra acknowledgement arrives correctly, the sender will not attempt to retransmit the lost frame, and the protocol will fail. In later protocols the acknowledgement frames will contain information to prevent just this sort of trouble. For the time being, the acknowledgement frames will just be dummies, and we will assume a strict alternation of sender and receiver.

Protocol 3 differs from its predecessors in that both sender and receiver have a variable whose value is remembered while the data link layer is in wait state. The sender remembers the sequence number of the next frame to send in *next_frame_to_send*; the receiver remembers the sequence number of the next frame expected in *frame_expected*. Each protocol has a short initialization phase before entering the infinite loop.

After transmitting a frame, the sender starts the timer running. If it was already running, it will be reset to allow another full timer interval. The time interval must be chosen to allow enough time for the frame to get to the receiver, for the receiver to process it in the worst case, and for the acknowledgement frame to propagate back to the sender. Only when that time interval has elapsed

```
/* Protocol 3 (par) allows unidirectional data flow over an unreliable channel. */

#define MAX_SEQ 1                          /* must be 1 for protocol 3 */
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"

void sender3(void)
{
 seq_nr next_frame_to_send;                /* seq number of next outgoing frame */
 frame s;                                  /* scratch variable */
 packet buffer;                            /* buffer for an outbound packet */
 event_type event;

 next_frame_to_send = 0;                   /* initialize outbound sequence numbers */
 from_network_layer(&buffer);              /* fetch first packet */
 while (true) {
     s.info = buffer;                      /* construct a frame for transmission */
     s.seq = next_frame_to_send;           /* insert sequence number in frame */
     to_physical_layer(&s);                /* send it on its way */
     start_timer(s.seq);                   /* if answer takes too long, time out */
     wait_for_event(&event);               /* frame_arrival, cksum_err, timeout */
     if (event == frame_arrival) {
           from_physical_layer(&s);        /* get the acknowledgement */
           if (s.ack == next_frame_to_send) {
                 from_network_layer(&buffer); /* get the next one to send */
                 inc(next_frame_to_send);  /* invert next_frame_to_send */
           }
     }
 }
}

void receiver3(void)
{
 seq_nr frame_expected;
 frame r, s;
 event_type event;

 frame_expected = 0;
 while (true) {
     wait_for_event(&event);               /* possibilities: frame_arrival, cksum_err */
     if (event == frame_arrival) {         /* a valid frame has arrived. */
           from_physical_layer(&r);        /* go get the newly arrived frame */
           if (r.seq == frame_expected) {  /* this is what we have been waiting for. */
                 to_network_layer(&r.info); /* pass the data to the network layer */
                 inc(frame_expected);      /* next time expect the other sequence nr */
           }
           s.ack = 1 – frame_expected;     /* tell which frame is being acked */
           to_physical_layer(&s);          /* none of the fields are used */
     }
 }
}
```

Fig. 3-11. A positive acknowledgement with retransmission protocol.

is it safe to assume that either the transmitted frame or its acknowledgement has been lost, and to send a duplicate.

After transmitting a frame and starting the timer, the sender waits for something exciting to happen. There are three possibilities: an acknowledgement frame arrives undamaged, a damaged acknowledgement frame staggers in, or the timer goes off. If a valid acknowledgement comes in, the sender fetches the next packet from its network layer and puts it in the buffer, overwriting the previous packet. It also advances the sequence number. If a damaged frame arrives or no frame at all arrives, neither the buffer nor the sequence number are changed, so that a duplicate can be sent.

When a valid frame arrives at the receiver, its sequence number is checked to see if it is a duplicate. If not, it is accepted, passed to the network layer, and an acknowledgement generated. Duplicates and damaged frames are not passed to the network layer.

3.4. SLIDING WINDOW PROTOCOLS

In the previous protocols, data frames were transmitted in one direction only. In most practical situations, there is a need for transmitting data in both directions. One way of achieving full-duplex data transmission is to have two separate communication channels and use each one for simplex data traffic (in different directions). If this is done, we have two separate physical circuits, each with a "forward" channel (for data) and a "reverse" channel (for acknowledgements). In both cases the bandwidth of the reverse channel is almost entirely wasted. In effect, the user is paying for two circuits but using only the capacity of one.

A better idea is to use the same circuit for data in both directions. After all, in protocols 2 and 3 it was already being used to transmit frames both ways, and the reverse channel has the same capacity as the forward channel. In this model the data frames from A to B are intermixed with the acknowledgement frames from A to B. By looking at the *kind* field in the header of an incoming frame, the receiver can tell whether the frame is data or acknowledgement.

Although interleaving data and control frames on the same circuit is an improvement over having two separate physical circuits, yet another improvement is possible. When a data frame arrives, instead of immediately sending a separate control frame, the receiver restrains itself and waits until the network layer passes it the next packet. The acknowledgement is attached to the outgoing data frame (using the *ack* field in the frame header). In effect, the acknowledgement gets a free ride on the next outgoing data frame. The technique of temporarily delaying outgoing acknowledgements so that they can be hooked onto the next outgoing data frame is known as **piggybacking**.

The principal advantage of using piggybacking over having distinct acknowledgement frames is a better use of the available channel bandwidth. The *ack* field in the frame header costs only a few bits, whereas a separate frame would need a header, the acknowledgement, and a checksum. In addition, fewer frames sent means fewer "frame arrived" interrupts, and perhaps fewer buffers in the receiver, depending on how the receiver's software is organized. In the next protocol to be examined, the piggyback field costs only 1 bit in the frame header. It rarely costs more than a few bits.

However, piggybacking introduces a complication not present with separate acknowledgements. How long should the data link layer wait for a packet onto which to piggyback the acknowledgement? If the data link layer waits longer than the sender's timeout period, the frame will be retransmitted, defeating the whole purpose of having acknowledgements. If the data link layer were an oracle and could foretell the future, it would know when the next network layer packet was going to come in, and could decide either to wait for it or send a separate acknowledgement immediately, depending on how long the projected wait was going to be. Of course, the data link layer cannot foretell the future, so it must resort to some ad hoc scheme, such as waiting a fixed number of milliseconds. If a new packet arrives quickly, the acknowledgement is piggybacked onto it; otherwise, if no new packet has arrived by the end of this time period, the data link layer just sends a separate acknowledgement frame.

In addition to its being only simplex, protocol 3 can fail under some peculiar conditions involving early timeout. It would be nicer to have a protocol that remained synchronized in the face of any combination of garbled frames, lost frames, and premature timeouts. The next three protocols are more robust and continue to function even under pathological conditions. All three belong to a class of protocols called **sliding window** protocols. The three differ among themselves in terms of efficiency, complexity, and buffer requirements, as discussed later.

In all sliding window protocols, each outbound frame contains a sequence number, ranging from 0 up to some maximum. The maximum is usually $2^n - 1$ so the sequence number fits nicely in an n-bit field. The stop-and-wait sliding window protocol uses $n = 1$, restricting the sequence numbers to 0 and 1, but more sophisticated versions can use arbitrary n.

The essence of all sliding window protocols is that at any instant of time, the sender maintains a set of sequence numbers corresponding to frames it is permitted to send. These frames are said to fall within the **sending window**. Similarly, the receiver also maintains a **receiving window** corresponding to the set of frames it is permitted to accept. The sender's window and the receiver's window need not have the same lower and upper limits, or even have the same size. In some protocols they are fixed in size, but in others they can grow or shrink as frames are sent and received.

Although these protocols give the data link layer more freedom about the order in which it may send and receive frames, we have most emphatically not dropped the requirement that the protocol must deliver packets to the destination network layer in the same order that they were passed to the data link layer on the sending machine. Nor have we changed the requirement that the physical communication channel is "wire-like," that is, it must deliver all frames in the order sent.

The sequence numbers within the sender's window represent frames sent but as yet not acknowledged. Whenever a new packet arrives from the network layer, it is given the next highest sequence number, and the upper edge of the window is advanced by one. When an acknowledgement comes in, the lower edge is advanced by one. In this way the window continuously maintains a list of unacknowledged frames.

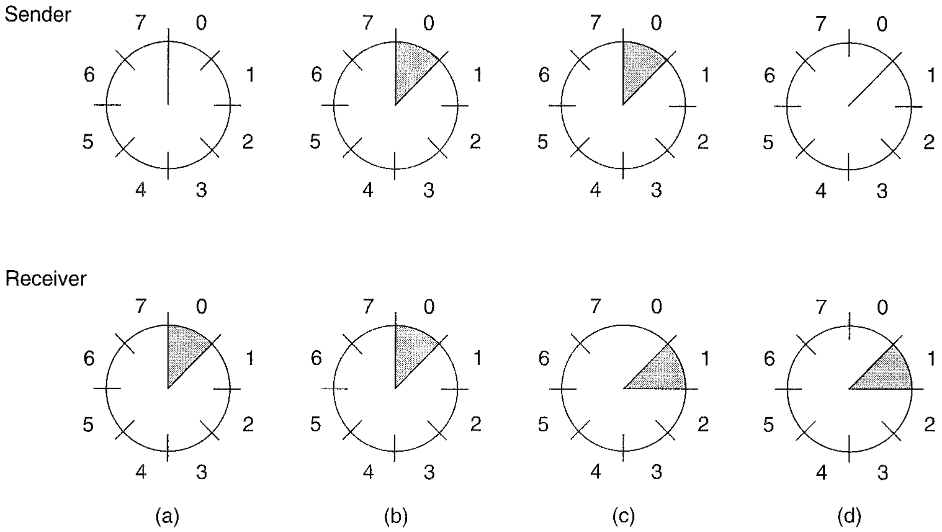


Fig. 3-12. A sliding window of size 1, with a 3-bit sequence number. (a) Initially. (b) After the first frame has been sent. (c) After the first frame has been received. (d) After the first acknowledgement has been received.

Since frames currently within the sender's window may ultimately be lost or damaged in transit, the sender must keep all these frames in its memory for possible retransmission. Thus if the maximum window size is *n*, the sender needs *n* buffers to hold the unacknowledged frames. If the window ever grows to its maximum size, the sending data link layer must forcibly shut off the network layer until another buffer becomes free.

The receiving data link layer's window corresponds to the frames it may accept. Any frame falling outside the window is discarded without comment. When a frame whose sequence number is equal to the lower edge of the window is received, it is passed to the network layer, an acknowledgement is generated, and the window is rotated by one. Unlike the sender's window, the receiver's

```
/* Protocol 4 (sliding window) is bidirectional and is more robust than protocol 3. */

#define MAX_SEQ 1                            /* must be 1 for protocol 4 */
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"

void protocol4 (void)
{
 seq_nr next_frame_to_send;                  /* 0 or 1 only */
 seq_nr frame_expected;                      /* 0 or 1 only */
 frame r, s;                                 /* scratch variables */
 packet buffer;                              /* current packet being sent */
 event_type event;

 next_frame_to_send = 0;                     /* next frame on the outbound stream */
 frame_expected = 0;                         /* number of frame arriving frame expected */
 from_network_layer(&buffer);                /* fetch a packet from the network layer */
 s.info = buffer;                            /* prepare to send the initial frame */
 s.seq = next_frame_to_send;                 /* insert sequence number into frame */
 s.ack = 1 – frame_expected;                 /* piggybacked ack */
 to_physical_layer(&s);                      /* transmit the frame */
 start_timer(s.seq);                         /* start the timer running */

 while (true) {
     wait_for_event(&event);                 /* frame_arrival, cksum_err, or timeout */
     if (event == frame_arrival) { /* a frame has arrived undamaged. */
          from_physical_layer(&r);           /* go get it */

          if (r.seq == frame_expected) {
               /* Handle inbound frame stream. */
               to_network_layer(&r.info);    /* pass packet to network layer */
               inc(frame_expected);          /* invert sequence number expected next */
          }

          if (r.ack == next_frame_to_send) { /* handle outbound frame stream. */
               from_network_layer(&buffer);       /* fetch new pkt from network layer */
               inc(next_frame_to_send);      /* invert sender's sequence number */
          }
     }
     s.info = buffer;                        /* construct outbound frame */
     s.seq = next_frame_to_send;             /* insert sequence number into it */
     s.ack = 1 – frame_expected;             /* seq number of last received frame */
     to_physical_layer(&s);                  /* transmit a frame */
     start_timer(s.seq);                     /* start the timer running */
 }
}
```

Fig. 3-13. A 1-bit sliding window protocol.

window always remains at its initial size. Note that a window size of 1 means that the data link layer only accepts frames in order, but for larger windows this is not so. The network layer, in contrast, is always fed data in the proper order, regardless of the data link layer's window size.

Figure 3-12 shows an example with a maximum window size of 1. Initially, no frames are outstanding, so the lower and upper edges of the sender's window are equal, but as time goes on, the situation progresses as shown.

3.4.1. A One Bit Sliding Window Protocol

Before tackling the general case, let us first examine a sliding window protocol with a maximum window size of 1. Such a protocol uses stop-and-wait, since the sender transmits a frame and waits for its acknowledgement before sending the next one.

Figure 3-13 depicts such a protocol. Like the others, it starts out by defining some variables. *Next_frame_to_send* tells which frame the sender is trying to send. Similarly, *frame_expected* tells which frame the receiver is expecting. In both cases, 0 and 1 are the only possibilities.

Normally, one of the two data link layers goes first. In other words, only one of the data link layer programs should contain the *to_physical_layer* and *start_timer* procedure calls outside the main loop. In the event both data link layers start off simultaneously, a peculiar situation arises, which is discussed later. The starting machine fetches the first packet from its network layer, builds a frame from it, and sends it. When this (or any) frame arrives, the receiving data link layer checks to see if it is a duplicate, just as in protocol 3. If the frame is the one expected, it is passed to the network layer and the receiver's window is slid up.

The acknowledgement field contains the number of the last frame received without error. If this number agrees with the sequence number of the frame the sender is trying to send, the sender knows it is done with the frame stored in *buffer* and can fetch the next packet from its network layer. If the sequence number disagrees, it must continue trying to send the same frame. Whenever a frame is received, a frame is also sent back.

Now let us examine protocol 4 to see how resilient it is to pathological scenarios. Assume that *A* is trying to send its frame 0 to *B* and that *B* is trying to send its frame 0 to *A*. Suppose that *A* sends a frame to *B*, but *A*'s timeout interval is a little too short. Consequently, *A* may time out repeatedly, sending a series of identical frames, all with $seq = 0$ and $ack = 1$.

When the first valid frame arrives at *B*, it will be accepted, and *frame_expected* will be set to 1. All the subsequent frames will be rejected, because *B* is now expecting frames with sequence number 1, not 0. Furthermore, since all the duplicates have $ack = 1$ and *B* is still waiting for an acknowledgement of 0, *B* will not fetch a new packet from its network layer.

After every rejected duplicate comes in, *B* sends *A* a frame containing *seq* = 0 and *ack* = 0. Eventually, one of these arrives correctly at *A*, causing *A* to begin sending the next packet. No combination of lost frames or premature timeouts can cause the protocol to deliver duplicate packets to either network layer, or to skip a packet, or to get into a deadlock.

However, a peculiar situation arises if both sides simultaneously send an initial packet. This synchronization difficulty is illustrated by Fig. 3-14. In part (a), the normal operation of the protocol is shown. In (b) the peculiarity is illustrated. If *B* waits for *A*'s first frame before sending one of its own, the sequence is as shown in (a), and every frame is accepted. However, if *A* and *B* simultaneously initiate communication, their first frames cross, and the data link layers then get into situation (b). In (a) each frame arrival brings a new packet for the network layer; there are no duplicates. In (b) half of the frames contain duplicates, even though there are no transmission errors. Similar situations can occur as a result of premature timeouts, even when one side clearly starts first. In fact, if multiple premature timeouts occur, frames may be sent three or more times.

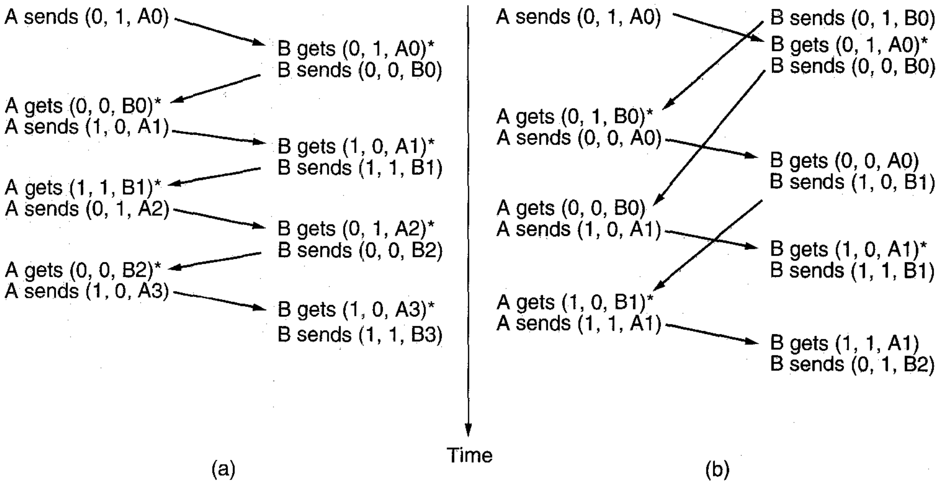


Fig. 3-14. Two scenarios for protocol 4. The notation is (seq, ack, packet number). An asterisk indicates where a network layer accepts a packet.

3.4.2. A Protocol Using Go Back n

Until now we have made the tacit assumption that the transmission time required for a frame to arrive at the receiver plus the transmission time for the acknowledgement to come back is negligible. Sometimes this assumption is clearly false. In these situations the long round-trip time can have important implications for the efficiency of the bandwidth utilization. As an example, consider a 50-kbps satellite channel with a 500-msec round-trip propagation delay. Let us imagine trying to use protocol 4 to send 1000-bit frames via the satellite. At $t = 0$ the sender starts sending the first frame. At $t = 20$ msec the frame has been completely sent. Not until $t = 270$ msec has the frame fully arrived at the receiver, and not until $t = 520$ msec has the acknowledgement arrived back at the sender, under the best of circumstances (no waiting in the receiver and a short acknowledgement frame). This means that the sender was blocked during 500/520 or 96 percent of the time (i.e., only 4 percent of the available bandwidth was used). Clearly, the combination of a long transit time, high bandwidth, and short frame length is disastrous in terms of efficiency.

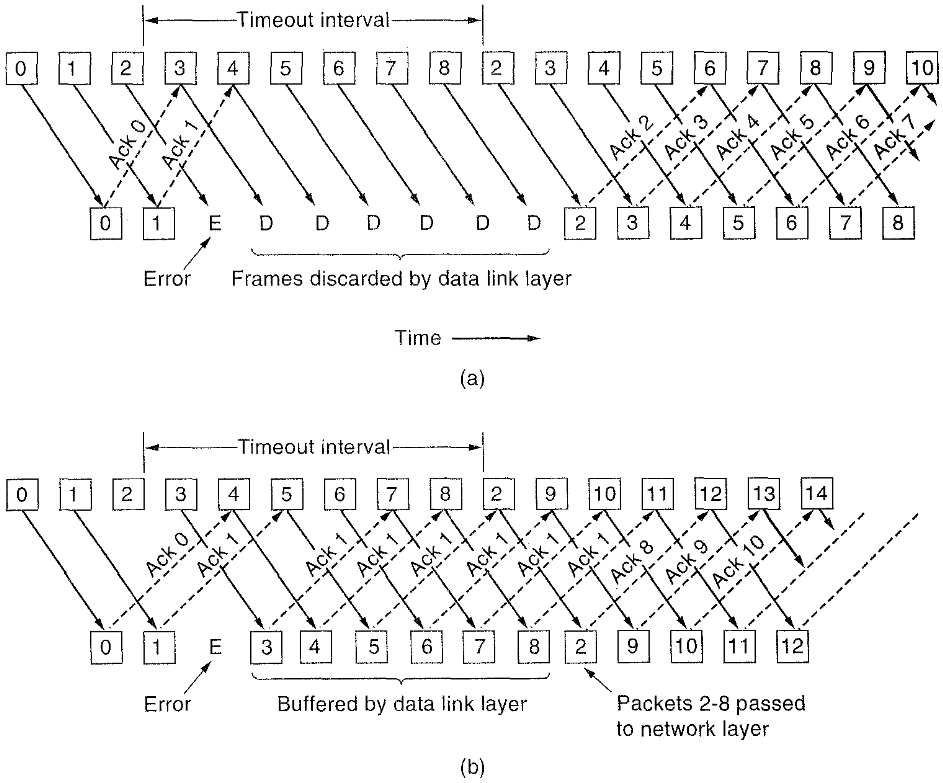


Fig. 3-15. (a) Effect of an error when the receiver window size is 1. (b) Effect of an error when the receiver window size is large.

The problem described above can be viewed as a consequence of the rule requiring a sender to wait for an acknowledgement before sending another frame. If we relax that restriction, much better efficiency can be achieved. Basically the solution lies in allowing the sender to transmit up to w frames before blocking, instead of just 1. With an appropriate choice of w the sender will be able to continuously transmit frames for a time equal to the round-trip transit time without filling up the window. In the example above, w should be at least 26. The sender begins sending frame 0 as before. By the time it has finished sending 26 frames, at $t = 520$, the acknowledgement for frame 0 will have just arrived. Thereafter, acknowledgements will arrive every 20 msec, so the sender always gets permission to continue just when it needs it. At all times, 25 or 26 unacknowledged frames are outstanding. Put in other terms, the sender's maximum window size is 26.

This technique is known as **pipelining**. If the channel capacity is b bits/sec, the frame size l bits, and the round-trip propagation time R sec, the time required to transmit a single frame is l/b sec. After the last bit of a data frame has been sent, there is a delay of $R/2$ before that bit arrives at the receiver, and another delay of at least $R/2$ for the acknowledgement to come back, for a total delay of R. In stop-and-wait the line is busy for l/b and idle for R, giving a line utilization of $l/(l + bR)$. If $l < bR$ the efficiency will be less than 50 percent. Since there is always a nonzero delay for the acknowledgement to propagate back, in principle pipelining can be used to keep the line busy during this interval, but if the interval is small, the additional complexity is not worth the trouble.

Pipelining frames over an unreliable communication channel raises some serious issues. First, what happens if a frame in the middle of a long stream is damaged or lost? Large numbers of succeeding frames will arrive at the receiver before the sender even finds out that anything is wrong. When a damaged frame arrives at the receiver, it obviously should be discarded, but what should the receiver do with all the correct frames following it? Remember that the receiving data link layer is obligated to hand packets to the network layer in sequence.

There are two basic approaches to dealing with errors in the presence of pipelining. One way, called **go back n**, is for the receiver simply to discard all subsequent frames, sending no acknowledgements for the discarded frames. This strategy corresponds to a receive window of size 1. In other words, the data link layer refuses to accept any frame except the next one it must give to the network layer. If the sender's window fills up before the timer runs out, the pipeline will begin to empty. Eventually, the sender will time out and retransmit all unacknowledged frames in order, starting with the damaged or lost one. This approach, shown in Fig. 3-15(a) can waste a lot of bandwidth if the error rate is high.

The other general strategy for handling errors when frames are pipelined, called **selective repeat**, is to have the receiving data link layer store all the correct frames following the bad one. When the sender finally notices that something is wrong, it just retransmits the one bad frame, not all its successors, as shown in Fig. 3-15(b). If the second try succeeds, the receiving data link layer will now have many correct frames in sequence, so they can all be handed off to the network layer quickly and the highest number acknowledged.

This strategy corresponds to a receiver window larger than 1. Any frame within the window may be accepted and buffered until all the preceding ones have

```
/* Protocol 5 (pipelining) allows multiple outstanding frames. The sender may transmit up
   to MAX_SEQ frames without waiting for an ack. In addition, unlike the previous protocols,
   the network layer is not assumed to have a new packet all the time. Instead, the
   network layer causes a network_layer_ready event when there is a packet to send. */

#define MAX_SEQ 7                    /* should be 2^n – 1 */
typedef enum {frame_arrival, cksum_err, timeout, network_layer_ready} event_type;
#include "protocol.h"

static boolean between(seq_nr a, seq_nr b, seq_nr c)
{
/* Return true if (a <=b < c circularly; false otherwise. */
  if (((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a)))
      return(true);
    else
      return(false);
}

static void send_data(seq_nr frame_nr, seq_nr frame_expected, packet buffer[])
{
/* Construct and send a data frame. */
  frame s;                           /* scratch variable */

  s.info = buffer[frame_nr];         /* insert packet into frame */
  s.seq = frame_nr;                  /* insert sequence number into frame */
  s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);/* piggyback ack */
  to_physical_layer(&s);             /* transmit the frame */
  start_timer(frame_nr);             /* start the timer running */
}

void protocol5(void)
{
  seq_nr next_frame_to_send;         /* MAX_SEQ > 1; used for outbound stream */
  seq_nr ack_expected;               /* oldest frame as yet unacknowledged */
  seq_nr frame_expected;             /* next frame expected on inbound stream */
  frame r;                           /* scratch variable */
  packet buffer[MAX_SEQ + 1];        /* buffers for the outbound stream */
  seq_nr nbuffered;                  /* # output buffers currently in use */
  seq_nr i;                          /* used to index into the buffer array */
  event_type event;

  enable_network_layer();            /* allow network_layer_ready events */
  ack_expected = 0;                  /* next ack expected inbound */
  next_frame_to_send = 0;            /* next frame going out */
  frame_expected = 0;                /* number of frame expected inbound */
  nbuffered = 0;                     /* initially no packets are buffered */
```

```
 while (true) {
   wait_for_event(&event);          /* four possibilities: see event_type above */

   switch(event) {
     case network_layer_ready:       /* the network layer has a packet to send */
          /* Accept, save, and transmit a new frame. */
          from_network_layer(&buffer[next_frame_to_send]); /* fetch new packet */
          nbuffered = nbuffered + 1;  /* expand the sender's window */
          send_data(next_frame_to_send, frame_expected, buffer);/* transmit the frame */
          inc(next_frame_to_send);    /* advance sender's upper window edge */
          break;

     case frame_arrival:              /* a data or control frame has arrived */
          from_physical_layer(&r);    /* get incoming frame from physical layer */

          if (r.seq == frame_expected) {
               /* Frames are accepted only in order. */
               to_network_layer(&r.info); /* pass packet to network layer */
               inc(frame_expected);  /* advance lower edge of receiver's window */
           }

           /* Ack n implies n – 1, n – 2, etc. Check for this. */
          while (between(ack_expected, r.ack, next_frame_to_send)) {
               /* Handle piggybacked ack. */
               nbuffered = nbuffered – 1; /* one frame fewer buffered */
               stop_timer(ack_expected); /* frame arrived intact; stop timer */
               inc(ack_expected);     /* contract sender's window */
          }
          break;

     case cksum_err: break;           /* just ignore bad frames */

     case timeout:                    /* trouble; retransmit all outstanding frames */
          next_frame_to_send = ack_expected;    /* start retransmitting here */
          for (i = 1; i <= nbuffered; i++) {
               send_data(next_frame_to_send, frame_expected, buffer);/* resend 1 frame */
               inc(next_frame_to_send);  /* prepare to send the next one */
          }

   }

   if (nbuffered < MAX_SEQ)
          enable_network_layer();
   else
          disable_network_layer();
 }
}
```

Fig. 3-16. A sliding window protocol using go back n.

been passed to the network layer. This approach can require large amounts of data link layer memory if the window is large.

These two alternative approaches are trade-offs between bandwidth and data link layer buffer space. Depending on which resource is more valuable, one or the other can be used. Figure 3-16 shows a pipelining protocol in which the receiving data link layer only accepts frames in order; frames following an error are discarded. In this protocol, for the first time, we have now dropped the assumption that the network layer always has an infinite supply of packets to send. When the network layer has a packet it wants to send, it can cause a *network_layer_ready* event to happen. However, in order to enforce the flow control rule of no more than *MAX_SEQ* unacknowledged frames outstanding at any time, the data link layer must be able to prohibit the network layer from bothering it with more work. The library procedures *enable_network_layer* and *disable_network_layer* perform this function.

Note that a maximum of *MAX_SEQ* frames and not *MAX_SEQ* + 1 frames may be outstanding at any instant, even though there are *MAX_SEQ* + 1 distinct sequence numbers: 0, 1, 2, . . . , *MAX_SEQ*. To see why this restriction is needed, consider the following scenario with *MAX_SEQ* = 7.

1. The sender sends frames 0 through 7.
2. A piggybacked acknowledgement for frame 7 eventually comes back to the sender.
3. The sender sends another eight frames, again with sequence numbers 0 through 7.
4. Now another piggybacked acknowledgement for frame 7 comes in.

The question is: Did all eight frames belonging to the second batch arrive successfully, or did all eight get lost (counting discards following an error as lost)? In both cases the receiver would be sending frame 7 as the acknowledgement. The sender has no way of telling. For this reason the maximum number of outstanding frames must be restricted to *MAX_SEQ*.

Although protocol 5 does not buffer the frames arriving after an error, it does not escape the problem of buffering altogether. Since a sender may have to retransmit all the unacknowledged frames at a future time, it must hang on to all transmitted frames until it knows for sure that they have been accepted by the receiver. When an acknowledgement comes in for frame *n*, frames *n* − 1, *n* − 2, and so on, are also automatically acknowledged. This property is especially important when some of the previous acknowledgement-bearing frames were lost or garbled. Whenever any acknowledgement comes in, the data link layer checks to see if any buffers can now be released. If buffers can be released (i.e., there is some room available in the window), a previously blocked network layer can now be allowed to cause more *network_layer_ready* events.

Because this protocol has multiple outstanding frames, it logically needs multiple timers, one per outstanding frame. Each frame times out independently of all the other ones. All of these timers can easily be simulated in software, using a single hardware clock that causes interrupts periodically. The pending timeouts form a linked list, with each node of the list telling how many clock ticks until the timer goes off, the frame being timed, and a pointer to the next node.

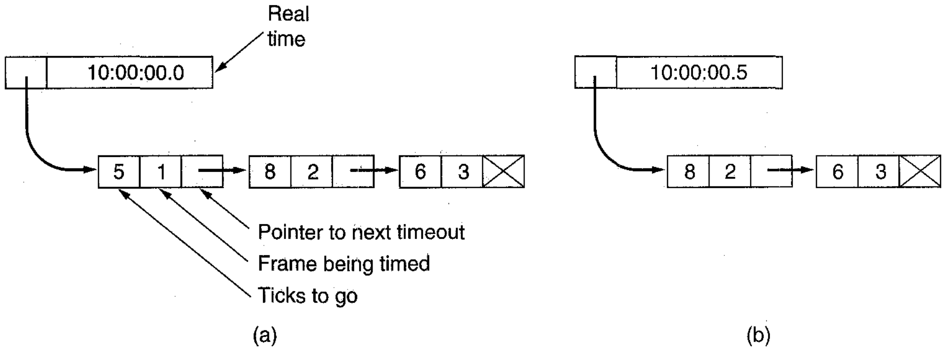


Fig. 3-17. Simulation of multiple timers in software.

As an illustration of how the timers could be implemented, consider the example of Fig. 3-17. Assume that the clock ticks once every 100 msec. Initially the real time is 10:00:00.0 and there are three timeouts pending, at 10:00:00.5, 10:00:01.3, and 10:00:01.9. Every time the hardware clock ticks, the real time is updated and the tick counter at the head of the list is decremented. When the tick counter becomes zero, a timeout is caused and the node removed from the list, as shown in Fig. 3-17(b). Although this organization requires the list to be scanned when *start_timer* or *stop_timer* is called, it does not require much work per tick. In protocol 5, both of these routines have been given a parameter, indicating which frame is to be timed.

3.4.3. A Protocol Using Selective Repeat

Protocol 5 works well if errors are rare, but if the line is poor it wastes a lot of bandwidth on retransmitted frames. An alternative strategy for handling errors is to allow the receiver to accept and buffer the frames following a damaged or lost one. Such a protocol does not discard frames merely because an earlier frame was damaged or lost.

In this protocol, both sender and receiver maintain a window of acceptable sequence numbers. The sender's window size starts out at 0 and grows to some predefined maximum, *MAX_SEQ*. The receiver's window, in contrast, is always fixed in size and equal to *MAX_SEQ*. The receiver has a buffer reserved for each sequence number within its window. Associated with each buffer is a bit (*arrived*) telling whether the buffer is full or empty. Whenever a frame arrives, its sequence number is checked by the function *between* to see if it falls within the window. If so, and if it has not already been received, it is accepted and stored. This action is taken without regard to whether or not it contains the next packet expected by the network layer. Of course, it must be kept within the data link layer and not passed to the network layer until all the lower numbered frames have already been delivered to the network layer in the correct order. A protocol using this algorithm is given in Fig. 3-18.

Nonsequential receive introduces certain problems not present in protocols in which frames are only accepted in order. We can illustrate the trouble most easily with an example. Suppose that we have a 3-bit sequence number, so that the sender is permitted to transmit up to seven frames before being required to wait for an acknowledgement. Initially the sender and receiver's windows are as shown in Fig. 3-19(a). The sender now transmits frames 0 through 6. The receiver's window allows it to accept any frame with sequence number between 0 and 6 inclusive. All seven frames arrive correctly, so the receiver acknowledges them and advance its window to allow receipt of 7, 0, 1, 2, 3, 4, or 5, as shown in Fig. 3-19(b). All seven buffers are marked empty.

It is at this point that disaster strikes in the form of a lightning bolt hitting the telephone pole and wiping out all the acknowledgements. The sender eventually times out and retransmits frame 0. When this frame arrives at the receiver, a check is made to see if it is within the receiver's window. Unfortunately, in Fig. 3-19(b) frame 0 is within the new window, so it will be accepted. The receiver sends a piggybacked acknowledgement for frame 6, since 0 through 6 have been received.

The sender is happy to learn that all its transmitted frames did actually arrive correctly, so it advances its window and immediately sends frames 7, 0, 1, 2, 3, 4, and 5. Frame 7 will be accepted by the receiver and its packet will be passed directly to the network layer. Immediately thereafter, the receiving data link layer checks to see if it has a valid frame 0 already, discovers that it does, and passes the embedded packet to the network layer. Consequently, the network layer gets an incorrect packet, and the protocol fails.

The essence of the problem is that after the receiver advanced its window, the new range of valid sequence numbers overlapped the old one. The following batch of frames might be either duplicates (if all the acknowledgements were lost) or new ones (if all the acknowledgements were received). The poor receiver has no way of distinguishing these two cases.

The way out of this dilemma lies in making sure that after the receiver has advanced its window, there is no overlap with the original window. To ensure that there is no overlap, the maximum window size should be at most half the range of the sequence numbers, as is done in Fig. 3-19(c) and Fig. 3-19(d). For example, if 4 bits are used for sequence numbers, these will range from 0 to 15.

Only eight unacknowledged frames should be outstanding at any instant. That way, if the receiver has just accepted frames 0 through 7 and advanced its window to permit acceptance of frames 8 through 15, it can unambiguously tell if subsequent frames are retransmissions (0 through 7) or new ones (8 through 15). In general, the window size for protocol 6 will be (*MAX_SEQ* + 1)/2.

An interesting question is: How many buffers must the receiver have? Under no conditions will it ever accept frames whose sequence numbers are below the lower edge of the window or frames whose sequence numbers are above the upper edge of the window. Consequently, the number of buffers needed is equal to the window size, not the range of sequence numbers. In the above example of a 4-bit sequence number, eight buffers, numbered 0 through 7, are needed. When frame i arrives, it is put in buffer i mod 8. Notice that although i and $(i + 8)$ mod 8 are "competing" for the same buffer, they are never within the window at the same time, because that would imply a window size of at least 9.

For the same reason, the number of timers needed is equal to the number of buffers, not the size of the sequence space. Effectively, there is a timer associated with each buffer. When the timer runs out, the contents of the buffer are retransmitted.

In protocol 5, there is an implicit assumption that the channel is heavily loaded. When a frame arrives, no acknowledgement is sent immediately. Instead, the acknowledgement is piggybacked onto the next outgoing data frame. If the reverse traffic is light, the acknowledgement will be held up for a long period of time. If there is a lot of traffic in one direction and no traffic in the other direction, only *MAX_SEQ* packets are sent, and then the protocol blocks.

In protocol 6 this problem is fixed. After an in-sequence data frame arrives, an auxiliary timer is started by *start_ack_timer*. If no reverse traffic has presented itself before this timer goes off, a separate acknowledgement frame is sent. An interrupt due to the auxiliary timer is called an *ack_timeout* event. With this arrangement, one-directional traffic flow is now possible, because the lack of reverse data frames onto which acknowledgements can be piggybacked is no longer an obstacle. Only one auxiliary timer exists, and if *start_ack_timer* is called while the timer is running, it is reset to a full acknowledgement timeout interval.

It is essential that the timeout associated with the auxiliary timer be appreciably shorter than the timer used for timing out data frames. This condition is required to make sure that the acknowledgement for a correctly received frame arrives before the sender times out and retransmits the frame.

Protocol 6 uses a more efficient strategy than protocol 5 for dealing with errors. Whenever the receiver has reason to suspect that an error has occurred, it sends a negative acknowledgement (NAK) frame back to the sender. Such a frame is a request for retransmission of the frame specified in the NAK. There are two cases when the receiver should be suspicious: a damaged frame has arrived or a frame other than the expected one arrived (potential lost frame). To avoid making

```
/* Protocol 6 (nonsequential receive) accepts frames out of order, but passes packets to the
   network layer in order. Associated with each outstanding frame is a timer. When the timer
   goes off, only that frame is retransmitted, not all the outstanding frames, as in protocol 5. */

#define MAX_SEQ 7                             /* should be 2^n - 1 */
#define NR_BUFS ((MAX_SEQ + 1)/2)
typedef enum {frame_arrival, cksum_err, timeout, network_layer_ready, ack_timeout} event_type;
#include "protocol.h"
boolean no_nak = true;                        /* no nak has been sent yet */
seq_nr oldest_frame = MAX_SEQ + 1;            /* initial value is only for the simulator */

static boolean between(seq_nr a, seq_nr b, seq_nr c)
{
/* Same as between in protocol5, but shorter and more obscure. */
  return ((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a));
}

static void send_frame(frame_kind fk, seq_nr frame_nr, seq_nr frame_expected, packet buffer[])
{
/* Construct and send a data, ack, or nak frame. */
  frame s;                                    /* scratch variable */

  s.kind = fk;                                /* kind == data, ack, or nak */
  if (fk == data) s.info = buffer[frame_nr % NR_BUFS];
  s.seq = frame_nr;                           /* only meaningful for data frames */
  s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);
  if (fk == nak) no_nak = false;              /* one nak per frame, please */
  to_physical_layer(&s);                      /* transmit the frame */
  if (fk == data) start_timer(frame_nr % NR_BUFS);
  stop_ack_timer();                           /* no need for separate ack frame */
}

void protocol6(void)
{
  seq_nr ack_expected;                        /* lower edge of sender's window */
  seq_nr next_frame_to_send;                  /* upper edge of sender's window + 1 */
  seq_nr frame_expected;                      /* lower edge of receiver's window */
  seq_nr too_far;                             /* upper edge of receiver's window + 1 */
  int i;                                      /* index into buffer pool */
  frame r;                                    /* scratch variable */
  packet out_buf[NR_BUFS];                    /* buffers for the outbound stream */
  packet in_buf[NR_BUFS];                     /* buffers for the inbound stream */
  boolean arrived[NR_BUFS];                   /* inbound bit map */
  seq_nr nbuffered;                           /* how many output buffers currently used */
  event_type event;

  enable_network_layer();                     /* initialize */
  ack_expected = 0;                           /* next ack expected on the inbound stream */
  next_frame_to_send = 0;                     /* number of next outgoing frame */
  frame_expected = 0;
  too_far = NR_BUFS;
  nbuffered = 0;                              /* initially no packets are buffered */
  for (i = 0; i < NR_BUFS; i++) arrived[i] = false;
```

```
    wait_for_event(&event);                       /* five possibilities: see event_type above */
    switch(event) {
      case network_layer_ready:                   /* accept, save, and transmit a new frame */
          nbuffered = nbuffered + 1;              /* expand the window */
          from_network_layer(&out_buf[next_frame_to_send % NR_BUFS]); /* fetch new packet */
          send_frame(data, next_frame_to_send, frame_expected, out_buf);/* transmit the frame */
          inc(next_frame_to_send);                /* advance upper window edge */
          break;

      case frame_arrival:                         /* a data or control frame has arrived */
          from_physical_layer(&r);                /* fetch incoming frame from physical layer */
          if (r.kind == data) {
              /* An undamaged frame has arrived. */
              if ((r.seq != frame_expected) && no_nak)
                send_frame(nak, 0, frame_expected, out_buf); else start_ack_timer();
              if (between(frame_expected, r.seq, too_far) && (arrived[r.seq%NR_BUFS] == false)) {
                  /* Frames may be accepted in any order. */
                  arrived[r.seq % NR_BUFS] = true;    /* mark buffer as full */
                  in_buf[r.seq % NR_BUFS] = r.info;   /* insert data into buffer */
                  while (arrived[frame_expected % NR_BUFS]) {
                      /* Pass frames and advance window. */
                      to_network_layer(&in_buf[frame_expected % NR_BUFS]);
                      no_nak = true;
                      arrived[frame_expected % NR_BUFS] = false;
                      inc(frame_expected);        /* advance lower edge of receiver's window */
                      inc(too_far);               /* advance upper edge of receiver's window */
                      start_ack_timer();          /* to see if a separate ack is needed */
                  }
              }
          }
          if((r.kind==nak) && between(ack_expected,(r.ack+1)%(MAX_SEQ+1),next_frame_to_send))
              send_frame(data, (r.ack+1) % (MAX_SEQ + 1), frame_expected, out_buf);

          while (between(ack_expected, r.ack, next_frame_to_send)) {
              nbuffered = nbuffered - 1;          /* handle piggybacked ack */
              stop_timer(ack_expected % NR_BUFS);     /* frame arrived intact */
              inc(ack_expected);                  /* advance lower edge of sender's window */
          }
          break;

      case cksum_err:
          if (no_nak) send_frame(nak, 0, frame_expected, out_buf);/* damaged frame */
          break;

      case timeout:
          send_frame(data, oldest_frame, frame_expected, out_buf);/* we timed out */
          break;

      case ack_timeout:
          send_frame(ack,0,frame_expected, out_buf);     /* ack timer expired; send ack */
    }
    if (nbuffered < NR_BUFS) enable_network_layer(); else disable_network_layer();
  }
}
```

Fig. 3-18. A sliding window protocol using selective repeat.

multiple requests for retransmission of the same lost frame, the receiver should keep track of whether a NAK has already been sent for a given frame. The variable *no_nak* in protocol 6 is true if no NAK has been sent yet for *frame_expected*. If the NAK gets mangled or lost, no real harm is done, since the sender will eventually time out and retransmit the missing frame anyway. If the wrong frame arrives after a NAK has been sent and lost, *no_nak* will be true and the auxiliary timer will be started. When it goes off, an ACK will be sent to resynchronize the sender to the receiver's current status.

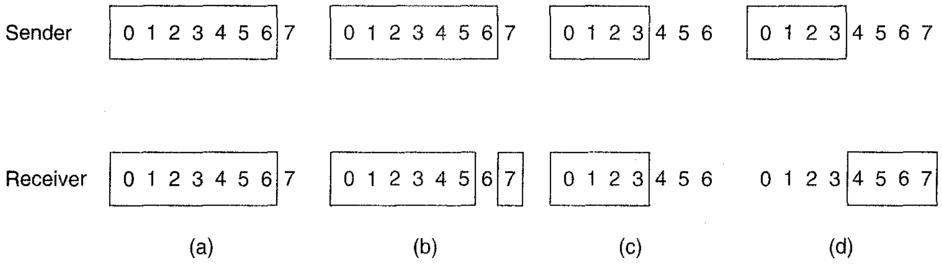


Fig. 3-19. (a) Initial situation with a window of size seven. (b) After seven frames have been sent and received but not acknowledged. (c) Initial situation with a window size of four. (d) After four frames have been sent and received but not acknowledged.

In some situations, the time required for a frame to propagate to the destination, be processed there, and have the acknowledgement come back is (nearly) constant. In these situations, the sender can adjust its timer to be just slightly larger than the normal time interval expected between sending a frame and receiving its acknowledgement. However, if this time is highly variable, the sender is faced with the choice of either setting the interval to a small value and risking unnecessary retransmissions, thus wasting bandwidth, or setting it to a large value, going idle for a long period after an error, thus also wasting bandwidth. If the reverse traffic is sporadic, the time before acknowledgement will be irregular, being shorter when there is reverse traffic and longer when there is not. Variable processing time within the receiver can also be a problem here. In general, whenever the standard deviation of the acknowledgement interval is small compared to the interval itself, the timer can be set "tight" and NAKs are not useful. Otherwise, the timer must be set "loose," and NAKs can appreciably speed up retransmission of lost or damaged frames.

Closely related to the matter of timeouts and NAKs is the question of determining which frame caused a timeout. In protocol 5 it is always *ack_expected*, because it is always the oldest. In protocol 6, there is no trivial way to determine who timed out. Suppose that frames 0 through 4 have been transmitted, meaning that the list of outstanding frames is 01234, in order from oldest to youngest. Now imagine that 0 times out, 5 (a new frame) is transmitted, 1 times out, 2 times out, and 6 (another new frame) is transmitted. At this point the list of outstanding frames is 3405126, from oldest to youngest. If all inbound traffic is lost for a while, the seven outstanding frames will time out in that order. To keep the example from getting even more complicated than it already is, we have not shown the timer administration. Instead, we just assume that the variable *oldest_frame* is set upon timeout to indicate which frame timed out.

3.5. PROTOCOL SPECIFICATION AND VERIFICATION

Realistic protocols, and the programs that implement them, are often quite complicated. Consequently, much research has been done trying to find formal, mathematical techniques for specifying and verifying protocols. In the following sections we will look at some models and techniques. Although we are looking at them in the context of the data link layer, they are also applicable to other layers.

3.5.1. Finite State Machine Models

A key concept used in many protocol models is the **finite state machine**. With this technique, each **protocol machine** (i.e., sender or receiver) is always in a specific state at every instant of time. Its state consists of all the values of its variables, including the program counter.

In most cases, a large number of states can be grouped together for purposes of analysis. For example, considering the receiver in protocol 3, we could abstract out from all the possible states two important ones: waiting for frame 0 or waiting for frame 1. All other states can be thought of as transient, just steps on the way to one of the main states. Typically, the states are chosen to be those instants that the protocol machine is waiting for the next event to happen [i.e., executing the procedure call *wait*(*event*) in our examples]. At this point the state of the protocol machine is completely determined by the states of its variables. The number of states is then 2^n, where n is the number of bits needed to represent all the variables combined.

The state of the complete system is the combination of all the states of the two protocol machines and the channel. The state of the channel is determined by its contents. Using protocol 3 again as an example, the channel has four possible states: a zero frame or a one frame moving from sender to receiver, an acknowledgement frame going the other way, or an empty channel. If we model the sender and receiver as each having two states, the complete system has 16 distinct states.

A word about the channel state is in order. The concept of a frame being "on the channel" is an abstraction, of course. What we really mean is that a frame has been partially transmitted, partially received, but not yet processed at the destination. A frame remains "on the channel" until the protocol machine executes *FromPhysicalLayer* and processes it.

From each state, there are zero or more possible **transitions** to other states. Transitions occur when some event happens. For a protocol machine a transition might occur when a frame is sent, when a frame arrives, when a timer goes off, when an interrupt occurs, etc. For the channel, typical events are insertion of a new frame onto the channel by a protocol machine, delivery of a frame to a protocol machine, or loss of a frame due to a noise burst. Given a complete description of the protocol machines and the channel characteristics, it is possible to draw a directed graph showing all the states as nodes and all the transitions as directed arcs.

One particular state is designated as the **initial state**. This state corresponds to the description of the system when it starts running, or some convenient starting place shortly thereafter. From the initial state, some, perhaps all, of the other states can be reached by a sequence of transitions. Using well-known techniques from graph theory (e.g., computing the transitive closure of a graph), it is possible to determine which states are reachable and which are not. This technique is called **reachability analysis** (Lin et al., 1987). This analysis can be helpful in determining if a protocol is correct or not.

Formally, a finite state machine model of a protocol can be regarded as a quadruple (S, M, I, T) where:

S is the set of states the processes and channel can be in.

M is the set of frames that can be exchanged over the channel.

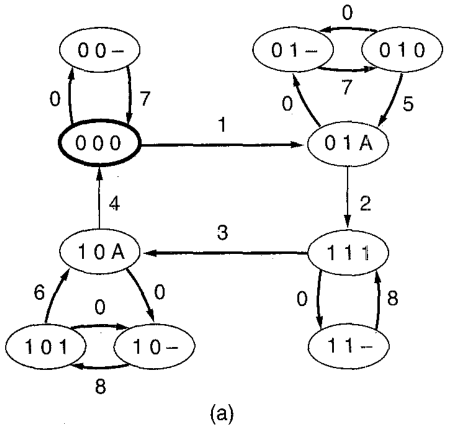
I is the set of initial states of the processes.

T is the set of transitions between states.

At the beginning of time, all processes are in their initial states. Then events begin to happen, such as frames becoming available for transmission or timers going off. Each event may cause one of the processes or the channel to take an action and switch to a new state. By carefully enumerating each possible successor to each state, one can build the reachability graph and analyze the protocol.

Reachability analysis can be used to detect a variety of errors in the protocol specification. For example, if it is possible for a certain frame to occur in a certain state and the finite state machine does not say what action should be taken, the specification is in error (incompleteness). If there exists a set of states from which there is no exit and from which no progress can be made (correct frames received), we have another error (deadlock). A less serious error is protocol specification that tells how to handle an event in a state in which the event cannot occur (extraneous transition). Other errors can also be detected.

As an example of a finite state machine model, consider Fig. 3-20(a). This graph corresponds to protocol 3 as described above: each protocol machine has two states and the channel has four states. A total of 16 states exist, not all of them reachable from the initial one. The unreachable ones are not shown in the figure. Each state is labeled by three characters, *XYZ*, where *X* is 0 or 1, corresponding to the frame the sender is trying to send; *Y* is also 0 or 1, corresponding to the frame the receiver expects, and *Z* is 0, 1, *A*, or empty (–), corresponding to the state of the channel. In this example the initial state has been chosen as (000). In other words, the sender has just sent frame 0, the receiver expects frame 0, and frame 0 is currently on the channel.



Transition	Who runs?	Frame accepted	Frame emitted	To network layer
0	–	(frame lost)		–
1	R	0	A	Yes
2	S	A	1	–
3	R	1	A	Yes
4	S	A	0	–
5	R	0	A	No
6	R	1	A	No
7	S	(timeout)	0	–
8	S	(timeout)	1	–

(b)

Fig. 3-20. (a) State diagram for protocol 3. (b) Transitions.

Nine kinds of transitions are shown in Fig. 3-20. Transition 0 consists of the channel losing its contents. Transition 1 consists of the channel correctly delivering packet 0 to the receiver, with the receiver then changing its state to expect frame 1 and emitting an acknowledgement. Transition 1 also corresponds to the receiver delivering packet 0 to the network layer. The other transitions are listed in Fig. 3-20(b). The arrival of a frame with a checksum error has not been shown because it does not change the state (in protocol 3).

During normal operation, transitions 1, 2, 3, and 4 are repeated in order over and over. In each cycle, two packets are delivered, bringing the sender back to the initial state of trying to send a new frame with sequence number 0. If the channel loses frame 0, it makes a transition from state (000) to state (00–). Eventually, the sender times out (transition 7) and the system moves back to (000). The loss of an acknowledgement is more complicated, requiring two transitions, 7 and 5, or 8 and 6, to repair the damage.

One of the properties that a protocol with a 1-bit sequence number must have is that no matter what sequence of events happens, the receiver never delivers two odd packets without an intervening even packet, and vice versa. From the graph of Fig. 3-20 we see that this requirement can be stated more formally as "there must not exist any paths from the initial state on which two occurrences of transition 1 occur without an occurrence of transition 3 between them, or vice versa." From the figure it can be seen that the protocol is correct in this respect.

Another, similar requirement is that there not be any paths on which the sender changes state twice (e.g., from 0 to 1 and back to 0) while the receiver state remains constant. Were such a path to exist, then in the corresponding sequence of events two frames would be irretrievably lost, without the receiver noticing. The packet sequence delivered would have an undetected gap of two packets in it.

Yet another important property of a protocol is the absence of deadlocks. A **deadlock** is a situation in which the protocol can make no more forward progress (i.e., deliver packets to the network layer) no matter what sequence of events happen. In terms of the graph model, a deadlock is characterized by the existence of a subset of states that is reachable from the initial state and which has two properties:

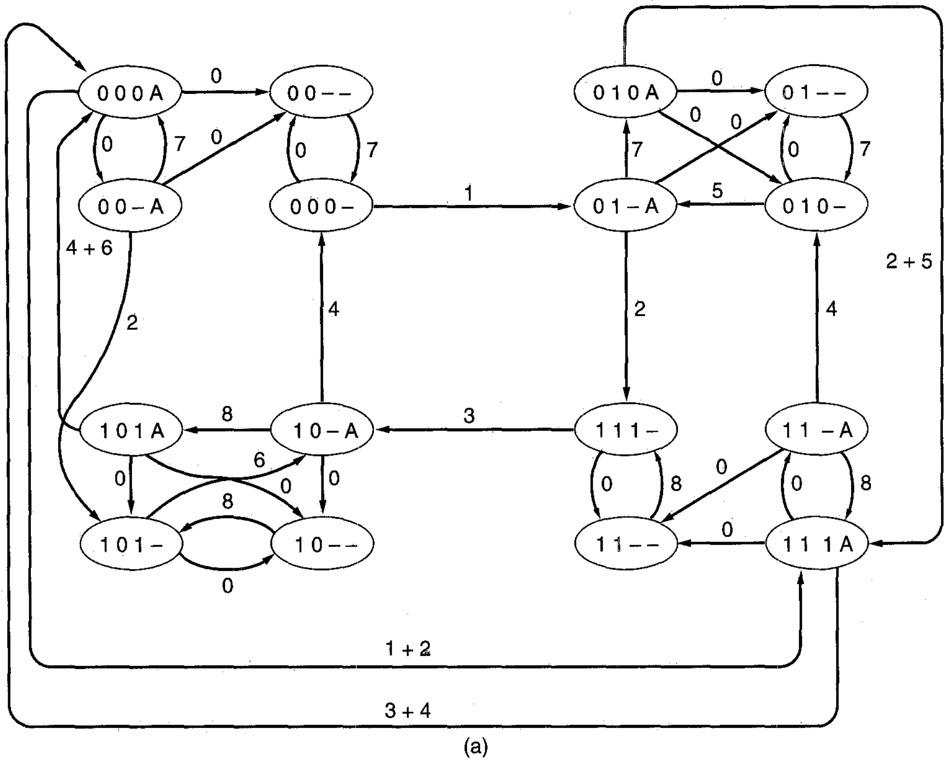
1. There is no transition out of the subset.
2. There are no transitions in the subset that cause forward progress.

Once in the deadlock situation, the protocol remains there forever. Again, it is easy to see from the graph that protocol 3 does not suffer from deadlocks.

Now let us consider a variation of protocol 3, one in which the half-duplex channel is replaced by a full-duplex channel. In Fig. 3-21 we show the states as the product of the states of the two protocol machines and the states of the two channels. Note that the forward channel now has three states: frame 0, frame 1, or empty, and the reverse channel has two states, *A* or empty. The transitions are the same as in Fig. 3-20(b), except that when a data frame and an acknowledgement are on the channel simultaneously, there is a slight peculiarity. The receiver cannot remove the data frame by itself, because that would entail having two acknowledgements on the channel at the same time, something not permitted in our model (although it is easy to devise a model that does allow it). Similarly, the sender cannot remove the acknowledgement, because that would entail emitting a second data frame before the first had been accepted. Consequently, both events must occur together, for example, the transition between state (000A) and state (111A), labeled as 1 + 2 in the figure.

In Fig. 3-21(a) there exist paths that cause the protocol to fail. In particular, there are paths in which the sender repeatedly fetches new packets, even though the previous ones have not been delivered correctly. The problem arises because it is now possible for the sender to time out and send a new frame without disturbing the acknowledgement on the reverse channel. When this acknowledgement arrives, it will be mistakenly regarded as referring to the current transmission and not the previous one.

One state sequence causing the protocol to fail is shown in Fig. 3-21(b). In



(0 0 0 –), (0 1 – A), (0 1 0 A), (1 1 1 A), (1 1 – A), (0 1 0 –), (0 1 – A), (1 1 1 –)

(b)

Fig. 3-21. (a) State graph for protocol 3 and a full-duplex channel. (b) Sequence of states causing the protocol to fail.

the fourth and sixth states of this sequence, the sender changes state, indicating that it fetches a new packet from the network layer, while the receiver does not change state, that is, does not deliver any packets to the network layer.

3.5.2. Petri Net Models

The finite state machine is not the only technique for formally specifying protocols. In this section we will describe another technique, the **Petri Net** (Danthine, 1980). A Petri net has four basic elements: places, transitions, arcs, and tokens. A **place** represents a state which (part of) the system may be in. Figure 3-22 shows a Petri net with two places, *A* and *B*, both shown as circles. The system is currently in state *A*, indicated by the **token** (heavy dot) in place *A*. A **transition** is indicated by a horizontal or vertical bar. Each transition has zero or more **input arcs**, coming from its input places, and zero or more **output arcs**, going to its output places.

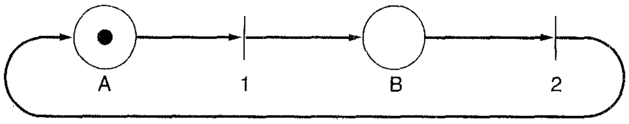


Fig. 3-22. A Petri net with two places and two transitions.

A transition is **enabled** if there is at least one input token in each of its input places. Any enabled transition may **fire** at will, removing one token from each input place and depositing a token in each output place. If the number of input arcs and output arcs differ, tokens will not be conserved. If two or more transitions are enabled, any one of them may fire. The choice of a transition to fire is indeterminate, which is why Petri nets are useful for modeling protocols. The Petri net of Fig. 3-22 is deterministic and can be used to model any two-phase process (e.g., the behavior of a baby: eat, sleep, eat, sleep, and so on). As with all modeling tools, unnecessary detail is suppressed.

Figure 3-23 gives the Petri net model of Fig. 3-21. Unlike the finite state machine model, there are no composite states here; the sender's state, channel state, and receiver's state are represented separately. Transitions 1 and 2 correspond to transmission of frame 0 by the sender, normally, and on a timeout respectively. Transitions 3 and 4 are analogous for frame 1. Transitions 5, 6, and 7 correspond to the loss of frame 0, an acknowledgement, and frame 1, respectively. Transitions 8 and 9 occur when a data frame with the wrong sequence number arrives at the receiver. Transitions 10 and 11 represent the arrival at the receiver of the next frame in sequence and its delivery to the network layer.

Petri nets can be used to detect protocol failures in a way similar to the use of finite state machines. For example, if some firing sequence included transition 10 twice without transition 11 intervening, the protocol would be incorrect. The concept of a deadlock in a Petri net is also similar to its finite state machine counterpart.

Petri nets can be represented in convenient algebraic form resembling a grammar. Each transition contributes one rule to the grammar. Each rule specifies the input and output places of the transition, for example, transition 1 in Fig. 3-23 is $BD \rightarrow AC$. The current state of the Petri net is represented as an unordered collection of places, each place represented in the collection as many times as it has tokens. Any rule all of whose left-hand side places are present, can be fired, removing those places from the current state, and adding its output places to the current state. The marking of Fig. 3-23 is *ACG*, so rule 10 ($CG \rightarrow DF$) can be applied but rule 3 ($AD \rightarrow BE$) cannot be applied.

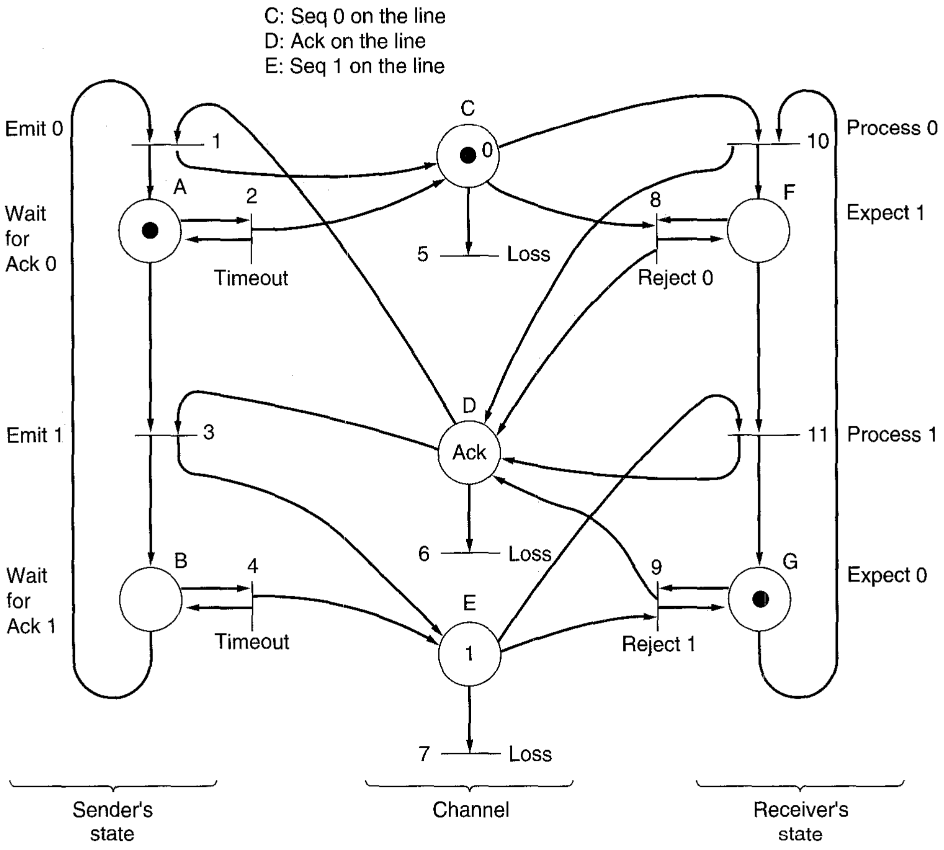


Fig. 3-23. A Petri net model for protocol 3.

3.6. EXAMPLE DATA LINK PROTOCOLS

In the following sections we will examine several widely-used data link protocols. The first one, HDLC, is common in X.25 and many other networks. After that, we will examine data link protocols used in the Internet and ATM networks, respectively. In subsequent chapters, we will also use the Internet and ATM as running examples as well.

3.6.1. HDLC—High-level Data Link Control

In this section we will examine a group of closely related protocols that are a bit old but are still heavily used in networks throughout the world. They are all derived from the data link protocol used in IBM's SNA, called **SDLC**

(**Synchronous Data Link Control**) protocol. After developing SDLC, IBM submitted it to ANSI and ISO for acceptance as U.S. and international standards, respectively. ANSI modified it to become **ADCCP** (**Advanced Data Communication Control Procedure**), and ISO modified it to become **HDLC** (**High-level Data Link Control**). CCITT then adopted and modified HDLC for its **LAP** (**Link Access Procedure**) as part of the X.25 network interface standard but later modified it again to **LAPB**, to make it more compatible with a later version of HDLC. The nice thing about standards is that you have so many to choose from. Furthermore, if you do not like any of them, you can just wait for next year's model.

All of these protocols are based on the same principles. All are bit-oriented, and all use bit stuffing for data transparency. They differ only in minor, but nevertheless irritating, ways. The discussion of bit-oriented protocols that follows is intended as a general introduction. For the specific details of any one protocol, please consult the appropriate definition.

All the bit-oriented protocols use the frame structure shown in Fig. 3-24. The *Address* field is primarily of importance on lines with multiple terminals, where it is used to identify one of the terminals. For point-to-point lines, it is sometimes used to distinguish commands from responses.

Bits	8	8	8	⩾0	16	8
	0 1 1 1 1 1 1 0	Address	Control	Data	Checksum	0 1 1 1 1 1 1 0

Fig. 3-24. Frame format for bit-oriented protocols.

The *Control* field is used for sequence numbers, acknowledgements, and other purposes, as discussed below.

The *Data* field may contain arbitrary information. It may be arbitrarily long, although the efficiency of the checksum falls off with increasing frame length due to the greater probability of multiple burst errors.

The *Checksum* field is a minor variation on the well-known cyclic redundancy code, using CRC-CCITT as the generator polynomial. The variation is to allow lost flag bytes to be detected.

The frame is delimited with another flag sequence (01111110). On idle point-to-point lines, flag sequences are transmitted continuously. The minimum frame contains three fields and totals 32 bits, excluding the flags on either end.

There are three kinds of frames: **Information**, **Supervisory**, and **Unnumbered**. The contents of the *Control* field for these three kinds are shown in Fig. 3-25. The protocol uses a sliding window, with a 3-bit sequence number. Up to seven unacknowledged frames may be outstanding at any instant. The *Seq* field in Fig. 3-25(a) is the frame sequence number. The *Next* field is a piggybacked acknowledgement. However, all the protocols adhere to the convention that instead of piggybacking the number of the last frame received correctly, they use the number of the first frame not received (i.e., the next frame expected). The choice of using the last frame received or the next frame expected is arbitrary; it does not matter which convention is used, provided that it is used consistently.

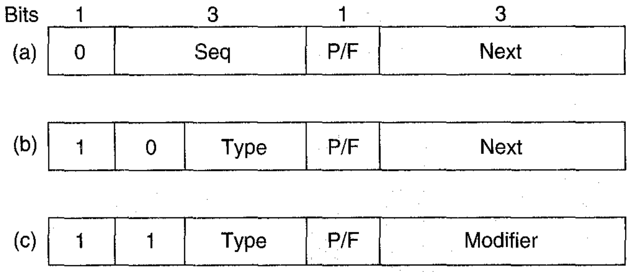


Fig. 3-25. Control field of (a) an information frame, (b) a supervisory frame, (c) an unnumbered frame.

The *P/F* bit stands for *Poll/Final*. It is used when a computer (or concentrator) is polling a group of terminals. When used as *P*, the computer is inviting the terminal to send data. All the frames sent by the terminal, except the final one, have the *P/F* bit set to *P*. The final one is set to *F*.

In some of the protocols, the *P/F* bit is used to force the other machine to send a Supervisory frame immediately rather than waiting for reverse traffic onto which to piggyback the window information. The bit also has some minor uses in connection with the Unnumbered frames.

The various kinds of Supervisory frames are distinguished by the *Type* field. Type 0 is an acknowledgement frame (officially called RECEIVE READY) used to indicate the next frame expected. This frame is used when there is no reverse traffic to use for piggybacking.

Type 1 is a negative acknowledgement frame (officially called REJECT). It is used to indicate that a transmission error has been detected. The *Next* field indicates the first frame in sequence not received correctly (i.e., the frame to be retransmitted). The sender is required to retransmit all outstanding frames starting at *Next*. This strategy is similar to our protocol 5 rather than our protocol 6.

Type 2 is RECEIVE NOT READY. It acknowledges all frames up to but not including *Next*, just as RECEIVE READY, but it tells the sender to stop sending. RECEIVE NOT READY is intended to signal certain temporary problems with the receiver, such as a shortage of buffers, and not as an alternative to the sliding window flow control. When the condition has been repaired, the receiver sends a RECEIVE READY, REJECT, or certain control frames.

Type 3 is the SELECTIVE REJECT. It calls for retransmission of only the frame specified. In this sense it is like our protocol 6 rather than 5 and is therefore most useful when the sender's window size is half the sequence space size, or less. Thus if a receiver wishes to buffer out of sequence frames for potential future use, it can force the retransmission of any specific frame using Selective Reject. HDLC and ADCCP allow this frame type, but SDLC and LAPB do not allow it (i.e., there is no Selective Reject), and type 3 frames are undefined.

The third class of frame is the Unnumbered frame. It is sometimes used for control purposes but can also be used to carry data when unreliable connectionless service is called for. The various bit-oriented protocols differ considerably here, in contrast with the other two kinds, where they are nearly identical. Five bits are available to indicate the frame type, but not all 32 possibilities are used.

All the protocols provide a command, DISC (DISConnect), that allows a machine to announce that it is going down (e.g., for preventive maintenance). They also have a command that allows a machine that has just come back on-line to announce its presence and force all the sequence numbers back to zero. This command is called SNRM (Set Normal Response Mode). Unfortunately, "Normal Response Mode" is anything but normal. It is an unbalanced (i.e., asymmetric) mode in which one end of the line is the master and the other the slave. SNRM dates from a time when data communication meant a dumb terminal talking to a computer, which clearly is asymmetric. To make the protocol more suitable when the two partners are equals, HDLC and LAPB have an additional command, SABM (Set Asynchronous Balanced Mode), which resets the line and declares both parties to be equals. They also have commands SABME and SNRME, which are the same as SABM and SNRM, respectively, except that they enable an extended frame format that uses 7-bit sequence numbers instead of 3-bit sequence numbers.

A third command provided by all the protocols is FRMR (FRaMe Reject), used to indicate that a frame with a correct checksum but impossible semantics arrived. Examples of impossible semantics are a type 3 Supervisory frame in LAPB, a frame shorter than 32 bits, an illegal control frame, and an acknowledgement of a frame that was outside the window, etc. FRMR frames contain a 24-bit data field telling what was wrong with the frame. The data include the control field of the bad frame, the window parameters, and a collection of bits used to signal specific errors.

Control frames may be lost or damaged, just like data frames, so they must be acknowledged too. A special control frame is provided for this purpose, called UA (Unnumbered Acknowledgement). Since only one control frame may be outstanding, there is never any ambiguity about which control frame is being acknowledged.

The remaining control frames deal with initialization, polling, and status reporting. There is also a control frame that may contain arbitrary information, UI (Unnumbered Information). These data are not passed to the network layer but are for the receiving data link layer itself.

Despite its widespread use, HDLC is far from perfect. A discussion of a variety of problems associated with it can be found in (Fiorini et al., 1995).

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

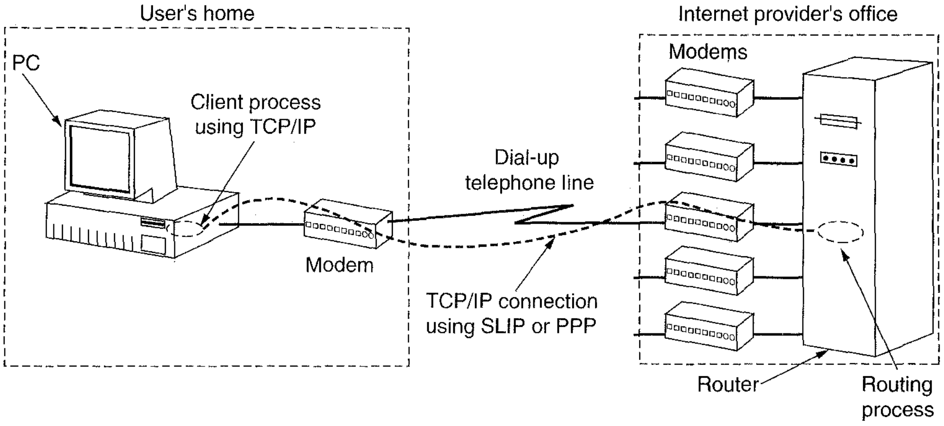


Fig. 3-26. A home personal computer acting as an Internet host.

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.