Network Protocols

ANDREW S. TANENBAUM

Wıskundıg Semınarıum, Vrıje Unıversıteıt, Amsterdam, The Netherlands

Durıng the last ten years, many computer networks have been designed, implemented, and put ınto service in the United States, Canada, Europe, Japan, and elsewhere. From the experıence obtaıned with these networks, certain key design principles have begun to emerge, prıncıples that can be used to desıgn new computer networks in a more structured way than has tradıtıonally been the case. Chıef among these principles is the notion of structurıng a network as a hıerarchy of layers, each one built upon the previous one. This paper is a tutorial about such network hierarchies, using the Reference Model of Open Systems Interconnectıon developed by the International Organization for Standardization as a guıde. Numerous examples are gıven to illustrate the principles.

Key Words and Phrases: computer network, data communicatıon, ISO OSI Reference Model, layered architecture, network, protocol

CR Categorıes· 1.3, 4.9, 6.9

INTRODUCTION

Ten years ago, only a handful of computer networks existed, mostly experimental networks built by research organizations. Today dozens of national and international networks and innumerable local networks operate on a commercial basis around the clock. From the beginning, many networks were designed hierarchically, as a series of layers, each one building on the one below. At first, each network design team started out by choosing its own set of layers. However, in the past few years, a consensus has begun to develop among network designers, a consensus embodied in the International Organization for Standardization's Reference Model of Open Systems Interconnection (ISO OSI). In this paper we present an informal introduction to computer networking using this model as a guide. A more thorough treatment of the ISO OSI model itself can be found in ZIMM80.

Before getting into the subject of network protocols, it is worth saying a few words about what we mean by a computer network. A computer network is a collection of computers, called *hosts*, that communicate with one another. The hosts may be large multiprogrammed mainframes or small personal computers. Networks can be classified as *local networks* or *long-haul networks.* The hosts on a local network are typically contained in a single building or campus and are connected by a high-bandwidth cable or other communication medium specifically designed for this purpose. Long-haul networks, in contrast, typically connect hosts in different cities using the public telephone network, an earth satellite, or both.

Local networks are nearly always completely owned by a single organization, whereas long-haul networks normally involve at least two organizations: the *carrier*, which operates the communication facility (telephone lines, microwave dishes, satellite, etc.), and the users, who own the hosts. This division of labor into (1) the provider of the communication facility and (2) the

CONTENTS

users of the communication facility has important ramifications for network architectures, as we shall see later.

The communication facility in a long-haul network is called the (*communication*) *subnet*, and often consists of a collection of minicomputers variously called *IMPs* (interface message processors), *nodes*, or *switches* connected by high-bandwidth leased telephone lines or a satellite. Figure 1 shows a network using telephone lines. Such a network is called a *point-to-point* or *store-and forward* network, as opposed to a *broadcast* network, such as a satellite network. The terms "host," "IMP," and "communication subnet" come from the U.S. Department of Defense's ARPANET, one of the first large-scale networks [McQu77]. We use this terminology generically because no consensus on nomenclature exists.

When the IMPs are connected by telephone lines, they are normally located on the carrier's premises, with each IMP servicing multiple hosts. To save on long-distance leased-line line charges, hosts and terminals are often funneled through remote concentrators. When the IMPs are connected by a satellite, the IMPs may be located on the customer's premises (e.g., on the roof). Local networks do not have IMPs; instead, each host has an interface card inserted into its backplane to control access to the network. This card is attached to the communication subnet, which is typically just a cable.

Although the ISO Reference Model can be used for both long-haul and local networks, it was designed primarily with the former in mind. Accordingly, in this paper we also treat both kinds of networks, but we emphasize slightly the long-haul variety, since issues such as routing and congestion control play a more prominent role in long-haul networks than in local networks.

In passing, we note that the subject of connecting distinct networks together is an increasingly important one, although it lies beyond the scope of this article. For an introduction to this subject see Bogg80 and Post80.

Protocols

As mentioned above, networks are almost always organized as a hierarchy of layers. Each layer performs a small set of closely related functions. The ISO Reference Model has seven layers:

(1) the physical layer,
(2) the data link layer,
(3) the network layer,
(4) the transport layer,
(5) the session layer,
(6) the presentation layer,
(7) the application layer,

as shown in Figure 2. All layers are present on the hosts, but only layers 1, 2, and 3 are present on the IMPs.

Each layer should be thought of as a program or process (possibly embedded in a hardware device) that communicates with

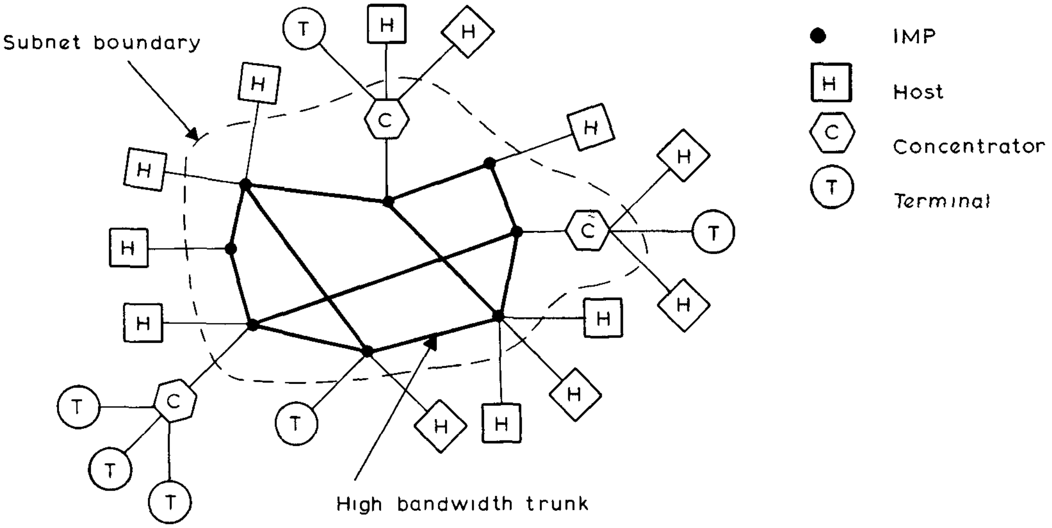


Figure 1. A typical point-to-point long-haul network.

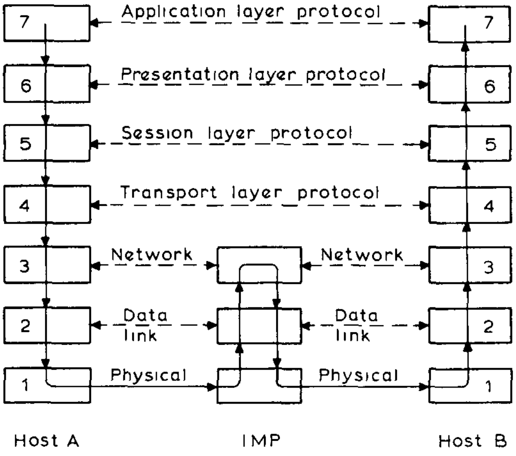


Figure 2. The seven-layer ISO Reference Model.

the corresponding process on another machine. In Figure 2, host layers 1, 2, and 3 think that they are communicating with their corresponding layers on the IMP, called *peers.* (In this example, hosts *A* and *B* are serviced by a common IMP; in general, multiple IMPs may intervene.) Layers 4-7, in contrast, communicate directly with their peer layers on the other host. The rules governing the layer *k* conversation are called the *layer k protocol.* The ISO model thus has seven protocols.

In reality, data are not transmitted horizontally, from machine to machine within a given layer, but are passed vertically down the layers of the sending machine and up the layers of the receiving machine. Only in layer 1 does actual intermachine communication occur. When an application program, running in layer 7 on host *A*, wants to send a message to the application in layer 7 on host *B*, it passes the message down to the presentation layer on its own machine. The presentation layer transforms the data, adds a layer 6 *header* containing control information used by the layer 6 protocol, and passes the resulting message down to the session layer. The session layer then adds its own header and passes the new message down to the transport layer. The complete path from layer 7 on host *A* to layer 7 on host *B* is shown in Figure 2 by the solid line. The boundary between adjacent layers is called an *interface.* The layers, interfaces, and protocols in a network form the *network architecture.*

No layer is aware of the header formats or protocols used by other layers. Layer *k* on the sending machine regards its job as getting the bits that come in from layer *k* + 1 over to the receiving machine somehow (using the services of the lower layers). It neither knows nor cares what the bits mean.

A three-layer analogy may be helpful in understanding how multilayer communication works. Consider the problem of the two talking philosophers. Philosopher 1 lives in an ivory tower in Kenya and speaks only Swahili. Philosopher 2 lives in a cave in India and speaks only Telugu. Nevertheless, Philosopher 1 wishes to convey his affection for *Oryctolagus cuniculus* to his Indian colleague (the philosophers are layer 3 peers). Since the philosophers speak different languages, each engages the services of a translator (layer 2 process) and an engineer (layer 1 process).

To convey his thoughts, Philosopher 1 passes his message, in Swahili, to his translator, across the 3/2 interface. The translator may convert it to English, French, Dutch, or some other language, depending only on the layer 2 protocol. The translator then hands his output to his engineer across the 2/1 interface for transmission. The physical mode of transmission may be telegram, telephone, computer network, or something else, depending only on the layer 1 protocol. When the Indian engineer receives the message, he passes it to his translator for rendition into Telugu. Finally, the Indian translator gives the message, in Telugu, to his philosopher.

This analogy illustrates three points. First, each person thinks of his communication as being primarily horizontal, with his peer (although in reality it is vertical, except in layer 1). For example, Philosopher 1 regards himself as conversing with Philosopher 2, even though his only physical communication is with translator 1. Second, actual communication is vertical, not horizontal, except in layer 1. Third, the three protocols are completely independent. The philosophers can switch the subject from rabbits to guinea pigs at will; the translators can switch from English to Dutch at will; the engineers can switch from telegram to telephone at will. The peers in any layer can change their protocol without affecting the other layers. It is for precisely this reason that networks are designed as a series of layers—to prevent changes in one part of the design (e.g., caused by technological advances) from requiring changes in other parts.

Overview of the ISO OSI Layers

The remainder of this article concerns the various layers in the ISO Reference Model, one section per layer. Before looking at the layers in detail, we first present a brief overview of each layer, to put the hierarchy in perspective.

The physical layer protocol is concerned with the transmission of a raw bit stream. Its protocol designers must decide how to represent 0's and 1's, how many microseconds a bit will last, whether transmission is full- or half-duplex, how the connection is set up and torn down, how many pins the network connector has, what each pin is used for, and other electrical, mechanical, and procedural details.

The data link layer converts an unreliable transmission channel into a reliable one for use by the network layer. The technique for doing so is to break up the raw bit stream into frames, each containing a checksum for detecting errors. (A *checksum* is a short integer that depends on all the bits in the frame so that a transmission error will probably change it and thus be detectable.) The data link protocol usually ensures that the sender of a data frame will repeatedly transmit the frame until it receives an acknowledgment frame from the receiver.

The network layer in a point-to-point network is primarily concerned with routing and the effects of poor routing, namely, congestion. In a broadcast network, routing is not an issue, since only one channel exists.

The task of the transport layer is to provide reliable host-to-host communication for use by the session layer. It must hide all the details of the communication subnet from the session layer, so that, for example, a point-to-point subnet can be replaced by a satellite link without affecting the session, presentation, or application layers. In effect, the transport layer shields the customer's portion of the network (layers 5-7) from the carrier's portion (layers 1-3).

The session layer is responsible for setting up, managing, and tearing down process-to-process connections, using the host-to-host service provided by the transport layer. It also handles certain aspects of synchronization and recovery.

The presentation layer performs generally useful transformations on the data to be sent, such as text compression. It also performs the conversions required to allow an interactive program to converse with any one of a set of incompatible intelligent terminals.

The content of the application layer is up to the users. Nevertheless, standard protocols for specific industries, such as airlines and banking, are likely to develop, although few exist now. For this reason we say no more about the application layer in this paper.

Although the ISO OSI Reference Model says nothing about how the layers are to be implemented, one possible configuration might have the physical layer in hardware, the data link layer in a special protocol chip, the network layer in a device driver, the transport and session layers in the operating system proper, the presentation layer in a set of library routines in the user's address space, and the application layer be the user's program.

At this point we have covered enough background material to say a little bit about the ISO OSI Reference Model itself. Basically, it is a framework for describing layered networks. It discusses the concept of layering in considerable detail, and introduces a uniform terminology for naming the various entities involved. Finally, it specifies the seven layers mentioned thus far, and for each layer gives its purpose, the services provided to the next higher layer, and a description of the functions that the layer must perform. The value of the model is that it provides a uniform nomenclature and a generally agreed upon way to split the various network activities into layers.

However, the ISO OSI Reference Model is *not* a protocol standard. By breaking a network's functions up into layers, it suggests places where protocol standards could be developed (physical layer protocols, data link layer protocols, and so on), but these standards themselves fall outside the domain of the model. With the model in hand, other organizations such as the Consultative Committee for International Telephony and Telegraphy (CCITT), the International Federation for Information Processing (IFIP), and the American National Standards Institute (ANSI) may develop specific protocol standards for the various layers. Although these standards may eventually be officially approved by ISO, such work is still in progress and, in any event, falls far outside the scope of the model.

As a final note, before plunging into the details of the various layers, we would like to point out that this article is about network protocols, with the ISO OSI Reference Model used as a guide; it is *not* an article about the model itself. We emphasize the communication algorithms and protocols themselves, a subject about which the Reference Model says nothing.

1. THE PHYSICAL LAYER

In this section we look at a variety of aspects related to the physical layer. Our emphasis is on the conceptual organization of the physical transmission facilities, not on the hardware details themselves. Point-to-point, satellite, and local networks are discussed. We conclude with a brief discussion of the X.21 physical layer protocol.

The function of the physical layer is to allow a host to send a raw bit stream into the network. The physical layer is in no way concerned with the way the bits are grouped into larger units, or what they mean. Nor does it rectify the problem of some bits being garbled by transmission errors. Recovery from such errors is up to the data link layer.

The communication subnet can be organized in one of two ways. In *circuit switching*, a fixed amount of transmission capacity (bandwidth) is reserved when the source initiates a conversation and released only when the conversation is over. The telephone system uses circuit switching. When someone calls a time-sharing service in a distant city, the connection is established after dialing and remains in force until one end hangs up. If the user goes out to lunch while still logged in, the connection remains intact and the charges continue to accumulate, even though the connection is actually idle.

With *packet switching*, in contrast, the user initially sets up a connection between his terminal or host and the nearest IMP, not the destination host. (We assume that the destination host also is connected to some IMP.) Whenever the user has data to send, he sends them to the IMP as a series of *packets*, typically 10–1000 bytes long.

Packets are routed from IMP to IMP within the subnet, until they get to the IMP which services the destination host. No circuits are reserved in advance within the subnet for the terminal-to-host connection (except the terminal-to-IMP and IMP-to-host circuits). Instead, the high-bandwidth IMP–IMP lines are dynamically shared among all the users on a demand basis; IMP–IMP bandwidth is only tied up when data are actually being transmitted.

Although the above discussion is cast in terms of a point-to-point network, the same considerations apply to broadcast channels. If a portion of the channel (e.g., one frequency band) is dedicated to a given conversation throughout its duration, without regard to actual usage, the network is circuit switched. If, however, the channel is dynamically requested, used, and released for every packet, the network is packet switched.

Circuit-switched networks are best suited to communication whose bandwidth requirements do not change much over time. Transmission of human speech is such an application, so it makes sense for the telephone network to be circuit switched. Terminal-to-computer and computer-to-computer traffic, however, is usually bursty. Most of the time there are no data to send, but once in a while a burst of data must be transmitted. For this reason, most computer networks use packet switching to avoid tying up expensive transmission facilities when they are not needed. However, in the future, all digital transmission systems will allow computers to dial a call, send the data, and hang up, all within a few milliseconds. If such systems become widespread, circuit switching may come back into favor.

1.1 The Telephone System

Since most existing long-haul networks use the telephone system for their transmission facilities, we shall briefly describe how the latter is organized. Most telephones are connected to a nearby telephone company switching office by a pair of copper wires known as a *local loop*. The switching offices themselves are connected by high-bandwidth *trunks* onto which thousands of unrelated calls are multiplexed. Although some trunks utilize copper wire, many utilize microwave relays, fiber optics, or wave guides as the transmission medium.

Because the bandwidth of the local loop is artificially limited to about 3000 Hz (hertz), it is difficult to transmit information over it by using, for example, +5 volts for a binary one and 0 volts for a binary zero. Such square wave signaling depends on high-frequency harmonics that are well above the 3000-Hz cutoff frequency. Only with very low date rates might enough information be below 3000 Hz to be intelligible. Instead, a device called a *modem* is inserted between the host and the telephone line. The input to the modem is pure digital data, but the output is a modulated sine wave with a base frequency of generally between 1000 and 2000 Hz. Since the modulated sine wave has fewer high-frequency components than the original square wave, it is affected less by the limited bandwidth.

A sine wave has three properties that can be modulated to transmit information: an amplitude, a frequency, and a phase. In amplitude modulation, two different amplitude values are used to represent 0 and 1. In frequency modulation, different frequencies are used for 0 and 1, but the amplitude is never varied. In phase modulation, neither the amplitude nor the frequency is varied, but the phase of the sine wave is abruptly switched to send data. In the most common encoding scheme, phase shifts of 45, 135, 225, and 315 degrees are used to send 00, 01, 10, and 11, respectively. In other words, each phase shift sends two bits. The three methods can be combined to increase the transmission capacity.

Many such transmission systems have been standardized and form an important class of physical layer protocols. Unfortunately, in many cases, the standards in the United States and Canada differ from those used by the rest of the world. For example, those ubiquitous 300-bit-per-second frequency modulation modems found near terminals around the world use different signaling frequencies in North America and Europe.

Probably the best known physical layer standard at present is RS-232-C, which specifies the meaning of each of the 25 pins on a terminal connector and the protocol governing their use. However, a new standard, RS-449, has been developed to replace this aging workhorse. RS-449 is upward compatible with RS-232-C but uses a 37-pin connector to accommodate the new signals. Unfortunately, 37 pins are insufficient, so users wishing to take advantage of all the features of RS-449 (notably the secondary channel) need a second 9-pin connector as well.

The transmission technology and protocols used on the interoffice trunks are different from those used on the local loop. In particular, digital rather than analog techniques are becoming increasingly widespread. The most common digital system is *pulse code modulation* (PCM), in which the analog signal coming in from the local loop is digitized by sampling it 8000 times per second. Eight bits (seven data and one control) are transmitted during each 125-μs (microsecond) sampling period. In North America, 24 such PCM channels are grouped together into 193-bit frames, with the last bit being used for synchronization. With 8000 193-bit frames per second, the gross data rate of this system, known as T1, is 1.544 Mbits/s (megabits per second). In Europe, the 1.544-Mbit/s PCM standard uses all 8 bits for data, with the 193rd bit (which is attached to the front rather than rear of the frame) used for signaling. Two different (and incompatible) 32-channel PCM standards running at 2.048 Mbits/s are also widely used outside North America. For more information about the telephone system see DAVI73 and DOLL78.

1.2 Communication Satellites

Although most existing long-haul networks use leased telephone circuits to connect the IMPs, satellite-based networks are becoming increasingly common. A communication satellite is a big repeater in the sky. Incoming signals are amplified and rebroadcast by a transponder on the satellite. The upward and downward signals use different frequencies to avoid interference. A typical communication satellite has 5–10 independent transponders, each with a capacity of about 50 Mbits/s.

Communication satellites are put into geosynchronous equatorial orbit at an altitude of 36,000 kilometers to make them appear stationary in the sky when viewed from the earth. Consequently, the ground station antenna can be pointed at the satellite when the antenna is installed and never moved. A moving satellite would require a much more expensive steerable antenna and would also have the disadvantage of being on the other side of the earth half the time. On the other hand, the great altitude required to achieve a 24-hour period implies an up-and-down propagation delay of 270 ms (milliseconds), which seriously affects the data link layer protocols and response time.

To avoid mutual interference, communication satellites using the 4/6-GHz (gigahertz) frequency band must be separated by an angle of 4 degrees as viewed from the earth. Since some orbit slots have been allocated by international agreement to television, military, and other use, the number of equatorial orbit slots available to data communication is limited. (As an aside, the allocation of orbit slots has been a political battleground, with every country, especially those in the Third World, asking for its fair share of slots for the purpose of renting them back to those countries able to launch satellites.) The 12/14-GHz band has also been allocated to data communication. At these frequencies, an orbit spacing of 1 degree is sufficient, providing four times as many slots. Unfortunately, because water is an excellent absorber of these short microwaves, multiple ground stations and elaborate switching are needed in order to avoid rain.

Three modes of operation have been proposed for satellite users. The most direct but most expensive mode is to put a complete ground station with antenna on the user's roof. This approach is already feasible for large multinational corporations and will become feasible for medium-sized ones as costs decline. The second approach is to put a small, cheap antenna on the user's roof to communicate with a shared satellite ground station on a nearby hill. The third approach is to access the ground station via a cable (e.g., a leased telephone circuit or even the same cable used for cable television).

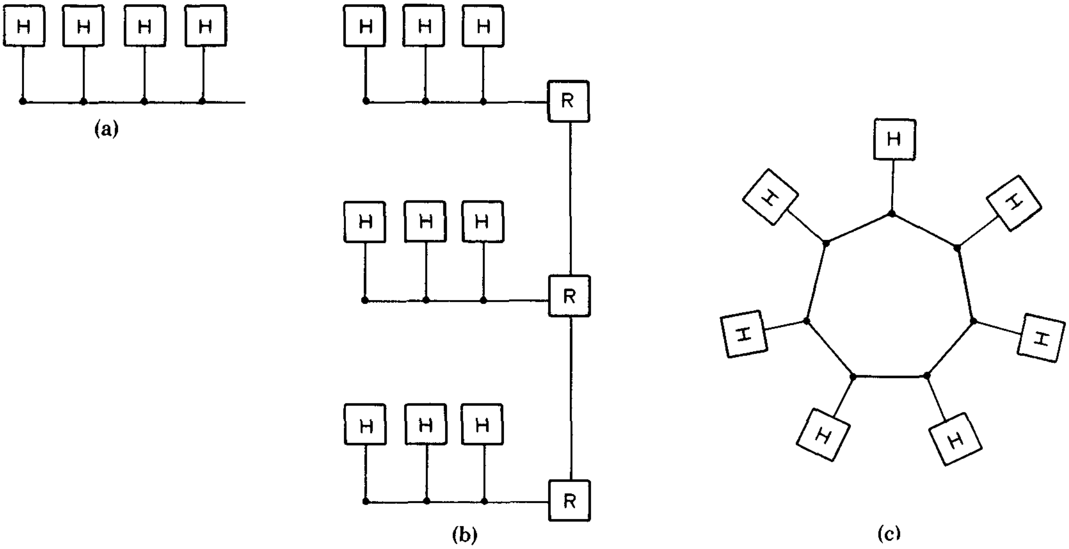


Figure 3. Local network topologies. (a) Linear cable with four hosts. (b) Segmented cable with repeaters and hosts. (c) Ring.

Physical layer satellite protocols typically have many PCM channels multiplexed on each transponder beam. Sometimes they are dedicated (circuit switched); at other times they are dynamically assigned as needed (packet switched). For more information about communication satellites see MART78.

1.3 Local Networks

In most local networks, the hosts are connected by a linear, tree-shaped, or ring-shaped cable, as shown in Figure 3. In Figure 3a, all hosts tap onto a common cable. In Figure 3b, multiple cables are used (e.g., one per floor of an office building), with repeaters connecting the segments. In Figure 3c, all hosts tap onto a unidirectional ring.

A widely imitated linear or tree-shaped local network is the Ethernet^{TM} network [METC76]. The proper term for this kind of network is CSMA/CD (Carrier Sense Multiple Access/Collision Detect), although many people incorrectly use the term "Ethernet" (which is a trademark of the Xerox Corporation) in a generic sense. In these networks, only one packet may be on the cable at any instant. The cable is known as the *ether*, after the luminiferous ether through which electromagnetic radiation was once alleged to propagate. The principle behind CSMA/CD is simple: when a host wishes to send a packet, it first listens to the ether to see if the ether is being used. If it is, the host waits until the current transmission finishes; if not, the host begins transmitting immediately.

The interface hardware must detect collisions caused by two hosts simultaneously starting a transmission. Collision detection is done using analog circuitry, in essence monitoring the ether to see if it agrees with the signal being transmitted. When a host interface (the analog of an IMP in this system, since the ether itself is totally passive) detects a collision, it informs the data link layer. The collision recovery action consists of aborting the current transmission, broadcasting a noise burst to make sure that everyone else detects the collision as well, waiting a random length of time, and then trying again. Collision detection is only feasible if the round-trip propagation delay is short compared to the packet transmission time, a condition that can be met with cable networks, but not, for example, with satellite networks.

Cable networks similar to the Xerox Ethernet network, but without the collision detect feature, also exist. Network designers can trade off the cost of collision detection circuitry against the time lost by not aborting colliding packets quickly.

Ring nets use a different principle: in effect, the whole ring is a giant circular shift

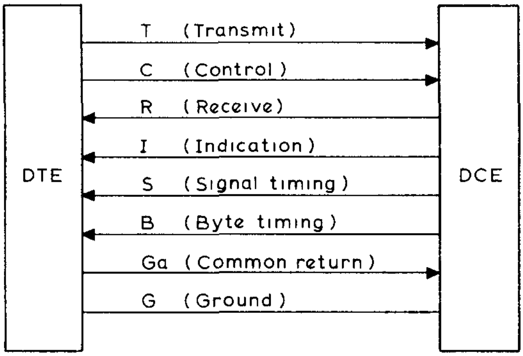


Figure 4. The DTE/DCE interface in X 21.

register. After each shift, the host interface can read or write the bit just shifted into it. Several different kinds of rings have been proposed [CLAR78, FARB72, FRAS75, LIU78, WILK79], differing primarily in their layer 2 organizations, which we describe later. Both CSMA/CD networks and rings typically operate at data rates of 1-10 Mbits/s. A substantial bibliography about local networks can be found in FREE80 and SHOC81.

1.4 An Example Physical Layer Protocol: X.21

At present, most physical layer standards, like RS-232-C and RS-449, utilize analog signaling. In the future, true digital interfaces will be needed. Recognizing this need, CCITT, the international standardization body for telephony, has developed a fully digital interface called *X.21*. X.21 is intended to be used to connect a host computer to a network. This connection remains established as long as the host wants to communicate with the network. Consequently, X.21 is a circuit-switched protocol, but host-host connections set up over the X.21 line may be either circuit switched or packet switched.

In X.21 terminology, the host is a *DTE* (Data Terminal Equipment) and the IMP is a *DCE* (Data Circuit-Terminating Equipment). The DTE-DCE interface consists of eight lines, as shown in Figure 4. The *S* line provides a clock signal to define bit boundaries. The (optional) *B* line provides a pulse every eighth bit, to allow byte alignment. The *C* and *I* lines are used for control signaling, analogous to the on-hook/off-hook signal on a telephone. The *T* and *R* lines are used for data and also for signaling.

To see how X.21 works, let us examine how a DTE calls another DTE, talks to it, and then hangs up. When the interface is idle, *T, R, C,* and *I* are all 1. The series of events is as follows (with a telephone analogy in parentheses):

(1) DTE drops *T* and *C* (DTE picks up phone).
(2) DCE sends "++++++ ··· +++" on *R* (DCE sends dial tone).
(3) DTE sends callee's address on *T* (DTE dials number).
(4) DCE sends call progress signals on *R* (phone rings).
(5) DCE drops *I* to 0 (callee answers phone).
(6) Full duplex data exchange on *T* and *R* (talk).
(7) DTE raises *C* to 1 (DTE says goodbye).
(8) DCE raises *I* to 1 (DCE says goodbye).
(9) DCE raises *R* to 1 (DCE hangs up).
(10) DTE raises *T* to 1 (DTE hangs up).

The call progress signals in Step 4 tell whether the call has been put through, and if not, why not. The shutdown procedure in Steps 7-10 operates in two phases. After either party has said goodbye, that party may not send more data but it must continue listening for incoming data. When both sides have said goodbye, they then hang up, returning the interface to idle state, with 1's on all four lines. RS-449 and X.21 are described in more detail in BERT80 and FOLT80.

2. THE DATA LINK LAYER

As we have seen, neither X.21, RS-232-C, nor any other physical layer protocol makes any attempt to detect or correct transmission errors. Nor do these protocols recognize the possibility that the receiver cannot accept data as fast as the sender can transmit them. Both of these problems are handled in the data link layer. In the following sections we first discuss the relevant principles and then we give an example of a widely used data link protocol, HDLC (High-Level Data Link Control). Following the HDLC example, we look at some data link protocols for satellite and local networks.

As mentioned earlier, the approach used in the data layer is to partition the raw physical layer bit stream into frames so each transmitted frame can be acknowledged if need be. An obvious question is: "How are frames delimited?" In other words, how can the receiver tell where one frame ends and the next one begins?

Three methods are in common use on long-haul networks: *character count, character stuffing,* and *bit stuffing.* With the first method, each frame begins with a fixed-format frame header that tells how many characters are contained in the frame. Thus, by simply counting characters, the receiver can detect the end of the current frame and the start of the following one. The method has the disadvantage of being overly sensitive to undetected transmission errors which affect the count field; it also has the disadvantage of enforcing a specific character size. Furthermore, lost characters wreak havoc with frame synchronization. Digital Equipment Corporation's DDCMP (Digital Data Communication Message Protocol) uses the character count method, but few other protocols do. Use of character counts to delimit frames is likely to diminish in the future.

The second method for delimiting frames, character stuffing, is to terminate each frame with a special "end-of-frame" character. The problem here is what to do with "end-of-frame" characters that accidently appear in the data (e.g., in the middle of a floating point number). The solution is to insert an "escape" character before every accidental "end-of-frame" character. Now what about accidental "escape" characters? These are rendered as two consecutive escapes. Although these conventions eliminate all ambiguity, they do so at the price of building a specific character code into the protocol. IBM's BISYNC (BInary SYNchronous Communication) protocol uses character stuffing, but, like all other such protocols, it is gradually becoming obsolete.

Modern data link protocols for long-haul networks all use bit stuffing, a technique in which frames are delimited by the bit pattern 01111110. Whenever five consecutive one bits appear in the data stream, a zero bit is "stuffed" into the bit stream (normally by hardware). Doing so prevents user data from interfering with framing, but does not impose any character size on the data.

On local networks, one can use any of the above methods, or a fourth method: detecting frames by the presence or absence of a signal on the cable. This method is much more direct, but it is not applicable to long-haul networks.

Virtually all data link protocols include a checksum in the frame header or trailer to detect, but not correct, errors. This approach has traditionally been used because error detection and retransmission requires fewer bits on the average than forward error correction (e.g., with a Hamming code). However, with the growing use of satellites, the long propagation delay makes forward error correction increasingly attractive.

A simple checksum algorithm is: compute the Exclusive OR of all the bytes or words as they are transmitted. This algorithm will detect all frames containing an odd number of bits in error, or a single error burst of length less than the checksum, and many other combinations. In practice, a more complex algorithm based on modulo 2 polynomial arithmetic is used [PETE61, SLOA75].

2.1 Stop-and-Wait Protocols

As a first example of a data link layer protocol, consider a host A wishing to send data to another host B over a perfectly reliable channel. At first glance you might think that A could just send at will. However, this idea does not work, since B may not be able to process the data as fast as they come in. If B had an infinite amount of buffer space, it could store the input for subsequent processing. Unfortunately, no host has infinite storage. Consequently, a mechanism is needed to throttle A into sending no faster than B can process the data. Such mechanisms are called *flow control* algorithms. The simplest one calls for A to send a frame and then wait for B to send explicit permission to send the next frame. This algorithm, called *stop-and-wait,* is widely used.

More elaborate protocols are needed for actual channels that make errors. An obvious extension to our basic protocol is to

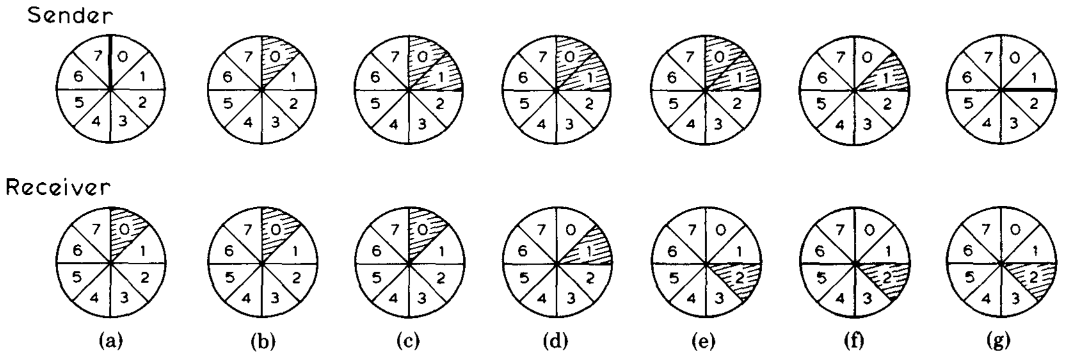


Figure 5. The sliding-window algorithm.

have A put a sequence number in each data link frame header and to have B put in each acknowledgment frame both a sequence number and a bit telling whether the checksum was correct or not. Whenever A received a negative acknowledgment frame (i.e., one announcing a checksum error), it could just repeat the frame.

Unfortunately, this protocol fails if either data or acknowledgment frames can be lost entirely in noise bursts. If a frame is lost, A will wait forever, creating a deadlock. Consequently, A must time out and repeat a frame if no acknowledgment is forthcoming within a reasonable period. Since each frame bears a sequence number, no harm is done if A has an itchy trigger finger and retransmits too quickly; however, some bandwidth is lost.

2.2 Sliding-Window Protocols

Stop-and-wait works well if the propagation time between the hosts is negligible. Consider, for a moment, how stop-and-wait works when 1000-bit frames are sent over a 1-Mbit/s satellite channel:

Time (ms)	Event
0	A starts sending the frame
1	Last bit sent, A starts to wait
270	First bit arrives at B
271	Last bit arrives at B
271	B sends a short acknowledgment
541	The acknowledgment arrives at A

For each millisecond of transmission, A has to wait 540 ms. The channel utilization is thus 1/541, or well below 1 percent. A better protocol is needed.

One such protocol is the *sliding-window* protocol, in which the sender is allowed to have multiple unacknowledged frames outstanding simultaneously. In this protocol, the sender has two variables, S_L and S_U, that tell which frames have been sent but not yet acknowledged. S_L is the lowest numbered frame sent but not yet acknowledged. The upper limit, S_U, is the first frame not yet sent. The current send window size is defined as $S_U - S_L$.

The receiver also has two variables, R_L and R_U, indicating that a frame with sequence number N may be accepted, provided that $R_L \leq N < R_U$. If $R_U - R_L = 1$, then the receiver has a window of size 1, that is, it only accepts frames in sequence. If the receiver's window is larger than 1, the receiver's data link layer may accept frames out of order, but normally it will just buffer such frames internally, so that it can pass frames to the network layer in order.

To keep sequence numbers from growing without bound, arithmetic is done modulo some power of 2. In the example of Figure 5, sequence numbers are recorded modulo 8. Initially (Figure 5a), $S_L = 0$, $S_U = 0$, $R_L = 0$, and $R_U = 1$ (receiver window size is 1 in this example). The current window is shown shaded in the figure. When the data link layer on the sending machine receives a frame to send (from the network layer), it sends the frame and advances the upper edge of its window by 1, as shown in Figure 5b. When it receives the next frame from the network layer, it sends the frame and advances the window again (Figure 5c). When the first frame arrives at the receiver, the receiver's window is rotated by advancing both edges (Figure 5d), and an acknowledgment is sent back. If frame 1 arrives at the receiver before the acknowledgment gets back to the sender, the state will be as shown in Figure 5e. When the first acknowledgment arrives, the lower edge of the sender's window is advanced (Figure 5f). Figure 5g shows the variables after both acknowledgments arrive.

As with stop-and-wait, the sliding-window protocol uses timeouts to recover from lost frames. The sender maintains a timer for each frame currently in its window. Whenever the lower edge of the window is advanced, the corresponding timer is stopped. Suppose, for example, that frames 0-4 are transmitted, but frame 1 is lost. The receiver will acknowledge frame 0, but discard frames 2-4 as they arrive, because they are outside the receive window (still size 1 in our example). Eventually, frames 1-4 will all time out and be retransmitted.

How many frames may our example sender have outstanding at any instant? The answer is seven, not eight, as might at first appear. To see why, consider the following scenario:

(1) The sender transmits frames 0-7.
(2) All eight frames arrive and are acknowledged.
(3) All eight acknowledgments are lost.
(4) The sender times out and retransmits the eight frames.
(5) The receiver unknowingly accepts the duplicates.

The problem occurs after Step 2, when the receiver's window has rotated all the way around and it is prepared to accept frame 0 again. Unfortunately, it cannot distinguish frame 9 from frame 0, so the stream of frames passed to the network layer will contain undetected duplicate frames.

The solution is to restrict the sender's window to seven outstanding frames. Then, after Step 2 above, the receiver will be expecting frame 7, and will reject all the duplicate frames, informing the sender after each rejection that it expects frame 7 next.

In the above example, whenever a frame is lost, the receiver is obligated to discard subsequent frames, even though they are correctly received. To avoid this inefficiency, we can allow the receiver's window to be greater than 1. Now let us look at the lost frame problem again, with both the sender's and receiver's windows of size 7. When frames 2-4 come in, the receiver keeps them internally. Eventually frame 1 times out and is retransmitted. The receiver replies to the correct receipt of frame 1 by saying that it expects frame 5 next, thereby implicitly acknowledging frames 2-4 and preventing their retransmission. With frames 1-4 now safely in hand, the data link layer can pass them to the network layer in sequence, thus completely shielding the latter from the lost frame and its recovery. This strategy is often called *selective repeat,* as opposed to the *go back n* strategy implied by a receiver window size of 1.

Unfortunately, even with the window settings used above, the protocol can fail. Consider the following scenario:

(1) The sender transmits frames 0-6.
(2) All frames arrive; the receiver's window is now 7, 0, 1, 2, 3, 4, 5.
(3) All seven acknowledgments are lost.
(4) The sender times out and retransmits frames 0-6.
(5) The receiver buffers frames 0-5 and says it wants frame 7 next.
(6) The sender transmits frames 7-13 (sequence numbers 7, 0, 1, 2, 3, 4, 5).
(7) The receiver accepts frame 7 but rejects frames 0-5 as duplicates.

At this point the receiver has frames 7, 0, 1, 2, 3, 4, and 5 buffered. It passes them all to the unsuspecting network layer. Consequently, undetected duplicates sneak through again. To prevent this, the window size must be restricted to not more than half the size of the sequence number space. With such a restriction, the receiver's window after having received a maximum batch of frames will not overlap what it was before having received the frames. Hence no ambiguity arises about whether a frame is a retransmission or an original.

2.3 An Example Data Link Protocol: HDLC

As an example of a data link protocol that is widely used, we now briefly look at HDLC (High-Level Data Link Control). HDLC has many brothers and sisters (e.g., SDLC, ADCCP, LAP, LAPB), each having

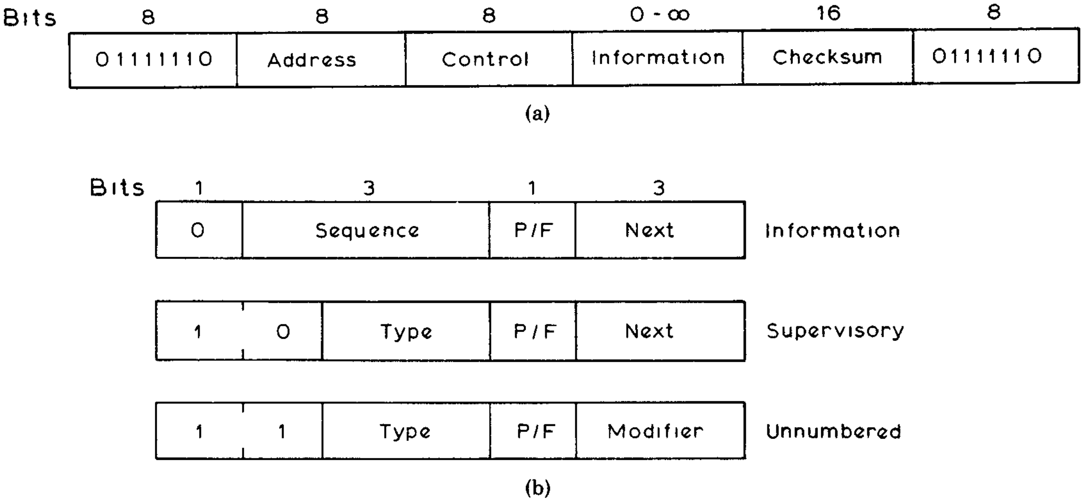


Figure 6. (a) The HDLC frame format. (b) The control byte for the three kinds of frames.

minor, but irritating, differences in the control frames. How this situation came about has to do with how certain large bureaucracies view certain other large bureaucracies, a stone best left unturned here.

HDLC and its friends all use bit stuffing for delimiting frames. Their format is shown in Figure 6a. The *Address* field is used for addressing on multipoint lines (lines connecting more than two computers). The *Control* field is different for each of the three classes of frames (see Figure 6b). In *Information* frames (i.e., ordinary data), the *Sequence* and *Next* fields contain the sequence number of the current frame and of the next frame expected, respectively. When *A* sends a frame to *B,* the *Sequence* field is the number of the frame being sent and the *Next* frame is an acknowledgment to *B* saying that *A* has correctly received all frames sent by *B* up to but not including *Next.* Attaching an acknowledgment field to an outgoing data frame is widely known as *piggybacking.* The practice saves bandwidth by requiring fewer frames. Reducing the number of frames *sent* also reduces the number of frames *received,* and hence reduces the number of I/O interrupts on the receiving machine.

When no reverse traffic is present on which to piggyback acknowledgments, a *Type* = 0 supervisory frame is used. The other types of supervisory frames are for negative acknowledgment, selective repeat, and receiver temporarily not ready. The *P/F* bit stands for *Poll/Final* and has miscellaneous uses, such as indicating polling frames on multipoint lines and the final frame in a sequence.

Unnumbered frames consist of a hodgepodge of control information and comprise the area of greatest difference between the various HDLC-like protocols. Most of these frames are used to initialize the line and to report certain abnormal conditions.

Although Figure 6 depicts HDLC as having a 3-bit sequence number, an alternate format with 7-bit sequence numbers also exists, for use on satellite or other channels where large windows are needed to keep the channel busy. Gelenbe et al. [GELE78] have constructed a mathematical model of HDLC that can be used to calculate the throughput as a function of window size.

2.4 Channel Allocation in Satellite Networks

At this point we switch from the data link layer of point-to-point networks to that of broadcast networks, in particular, satellite and local networks. Broadcast networks are characterized by having a single channel that is dynamically requested and released by hosts for every packet sent. A protocol is needed for determining who may use the channel when, how to prevent channel overload, and so on. These problems do not occur in point-to-point networks. On the other hand, since every host receives every packet, broadcast networks usually do not have to make any routing decisions. Thus the main function of the network layer is not relevant.

As a consequence of these fundamental differences, it is not really clear where the channel-access protocol should be placed in the ISO OSI Reference Model, which does not mention the issue at all. It could be put in the data link layer, since it deals with getting packets from one machine to the next, but it could equally well be put in the network layer, since it also concerns getting packets from the source host to the destination host. Another argument for putting it in the network layer is that the main task of the access protocol is to avoid congestion on the channel, and congestion control is specifically a network layer function. Last, in most broadcast networks the transport layer is built directly on top of this protocol, or in some cases on top of an internetwork protocol, something lacking in the ISO OSI Reference Model. Nevertheless, we treat the subject as part of the data link layer because the IEEE local network standards committee (802) is probably going to put it there. By analogy, the contention resolution protocol for satellite channels also belongs in the data link layer.

A satellite link can be operated like a terrestrial point-to-point link, providing dedicated bandwidth for each user by time-division or frequency-division multiplexing. In this mode the data link protocols are the same as in point-to-point networks, albeit with longer timeouts to account for the longer propagation delay.

Another mode of operation, however, is to dynamically assign the channel among the numerous competing users. Since their only method of communication is via the channel itself, the protocol used for allocating the channel is nontrivial. Abramson [ABRA70] and Roberts [ROBE73] have devised a method, known as *slotted ALOHA*, that has some interesting properties. In their approach, time is slotted into units of a (fixed-length) packet. During each interval, a host having a packet to send can either send or refrain from sending. If no hosts use a given slot, the slot is just wasted. If one host uses a slot, a successful transmission occurs. If two or more hosts try to use the same slot, a collision occurs and the slot is also wasted. Note that with satellites the hosts do not discover the collision until 270 ms after they start sending the packets. Owing to this long delay, the collision detection principle from CSMA/CD is not applicable here. Instead, after detecting a collision, each host waits a random number of slots and tries again.

Clearly, if few hosts have packets to send, few collisions will occur and the success rate will be high. If, on the other hand, many hosts have packets to send, many collisions will occur and the success rate will be low. In both cases the throughput will be low: in the first case because of lack of offered traffic, in the second case because of collisions. Hence the throughput versus offered traffic curve starts out low, peaks, and then falls again. Abramson [ABRA73] showed that the peak occurs when the mean offered traffic is one packet per slot, which yields a throughput of $1/e$ or about 0.37 packets per slot. Hence the best one can hope for with slotted ALOHA is a 37 percent channel utilization.

Slotted ALOHA has another problem, in addition to the low throughput: stability. Suppose that an ALOHA system has many hosts. By accident, during one slot k hosts transmit and collide. After detecting the collision, each host decides to retransmit during the next slot with probability p (a parameter of the system). In other words, each host picks a random number between 0 and 1. If the number is less than p, it transmits; otherwise it waits until the next slot to pick another random number.

If $kp \gg 1$, many hosts will retransmit during each succeeding slot and practically nothing will get through. Worse yet, these retransmissions will compete with new packets from other hosts, increasing the number of hosts trying to use the channel, which just makes the problem worse. Pretty soon all hosts will be trying to send and the throughput will approach zero, collapsing the system permanently.

The trick to avoid collapse is to set the parameter p low enough that $kp < 1$ for the

k values expected. However, the lower p is, the longer it takes even to attempt retransmission, let alone succeed. Hence a low value of p leads to a stable system, but only at the price of long delay times.

One way to set p is to use a default value on the first retransmission, say 0.5, on the assumption that two hosts are involved in the collision. On each subsequent collision, halve p. Gerla and Kleinrock [GERL77] have another proposal; they suggest that each host monitor the channel all the time, just to measure the collision rate. When the collision rate is low, the hosts can set p high; when the collision rate is high, the hosts can set p low to minimize collisions and get rid of the backlog, albeit slowly.

A completely different way to avoid collisions is to attempt to schedule the slots in advance rather than have continuous competition for them. Crowther et al. [CROW73] proposed grouping slots into n-slot time slices, with the time slice longer than the propagation delay. In their system, contention is used initially, as described above. Once a host has captured (i.e., successfully used) a slot, it is entitled to use the same slot position in the next slice, forbidding all other hosts from trying to use it. This algorithm makes it possible for a host to transmit a long file without too much pain. If a host no longer needs a slot position, it sets a bit in the packet header that permits other users to contend for the slot the next time around.

Roberts [ROBE73] also proposed a method of reducing contention. His proposal also groups slots into time slices. One slot per slice is divided into minislots and used for reserving regular slots. To send a packet, a host must first compete for a minislot. Since all hosts see the results of the minislot contention, they can all keep track of how long the queue is and hence know who gets to send when. In effect, the use of minislots greatly reduces the amount of time wasted on a collision (like the CSMA/CD rule about aborting collisions as soon as they are detected).

2.5 Channel Allocation in Local Networks

As mentioned earlier, when a CSMA/CD host detects a collision, it jams the channel, aborts the current packet, waits a random time, and tries again. How long should it wait? Metcalfe and Boggs [METC76] decided to use a default maximum time interval on the first collision, with the actual waiting time being picked by multiplying a random number in the range 0.0–1.0 by the maximum time interval. On each successive collision the maximum time interval is doubled and a new random number is generated. They called their algorithm *binary exponential backoff.*

Various other algorithms have been proposed for CSMA/CD, including some that prevent all collisions. For example, Chlamtac [CHLA76], Chlamtac et al. [CHLA79], and Scholl [SCHO76] have suggested slotting time into intervals equal to the channel acquisition time (the round-trip propagation delay). After a successful transmission by host n, the next bit slot is then reserved for host $(n + 1)$ (modulo the number of hosts). If the indicated host does not claim its right to use the channel, the next host gets a chance during the succeeding bit slot, and so on. In effect, a virtual baton is passed from host to host, with hosts only allowed to transmit when holding the baton.

Rothauser and Wild [ROTH77] have also proposed a collision-free CSMA/CD protocol. To illustrate their suggestion, we shall assume that there are 1024 hosts, numbered from 0 to 1023 (in binary, although other radices can also be used). After a successful transmission, ten bit slots will be used to determine who goes next. Each host attempts to broadcast its 10-bit number in the ten slots, subject to the rule that as soon as a host realizes that a higher numbered host wants the channel, it must stop trying. If, for example, the first three bits are 011, then some host in the range 368–511 wants the channel, and so hosts below 368 must desist from competing on the current round. No host above 511 wants the channel, as evidenced by the leading 0 bit. In effect, the channel is allocated to the highest numbered contender. Since this system gives high-numbered stations an advantage, it is desirable to make the host numbers virtual, rotating them one position after each successful transmission.

Protocols that allow only a limited number of collisions have also been proposed [CAPE79a, CAPE79b, KLEI78]. Capetanakis'

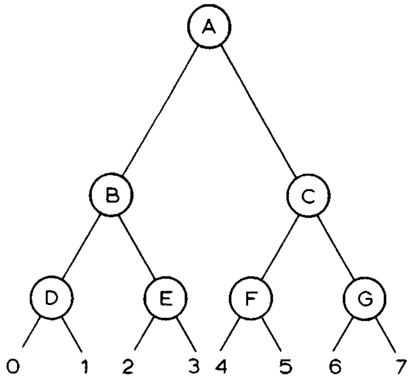


Figure 7. Eight machines organized in Capetanakis' (virtual) tree.

idea is illustrated in Figure 7 for a system with eight hosts. Initially, all hosts may compete. If a collision occurs, only those hosts under node *B* of the tree, namely, 0, 1, 2, and 3, may compete. If another collision occurs, the only descendants of node *D* may try, and so forth. As an example, suppose hosts 2, 3, and 4 all want the channel. After initial collisions for *A* (2, 3, and 4) and *B* (2 and 3), it will be node *D*'s turn and the channel will lie idle. Next comes node *E* and another collision. Then 2 and 3 each get a private slot, followed by *C*. At low load, the algorithm allows pure contention, but under high load it walks the tree looking for hosts that want to send.

Although more could be said about CSMA/CD protocols, we now turn our attention to ring networks. In one of the best-known rings [FARB72], an 8-bit token circulates around the ring when there is no traffic. When a host wants to transmit, it must first capture and destroy the token. Having done so, it may send its packet. When it is finished, it must put the token back, giving the next host downstream a chance to seize it.

If the token is ever lost (e.g., as a result of a ring interface malfunctioning), some mechanism is needed to regenerate it. One possibility is that each host wishing to send must monitor the ring. Having failed to see a token within the worst case interval—namely, all other hosts sending a maximum-length packet—the host generates a new token itself. However, with a little bad luck, two hosts might generate tokens simultaneously. Hence, it appears that token recovery in a ring net is similar to contention in CSMA/CD in systems. Clark et al. [CLAR78] have taken this observation to its logical conclusion and proposed a contention ring that is a hybrid of the token ring and CSMA/CD.

Yet another type of ring is exemplified by the Cambridge Ring [NEED79, WILK79]. This 10-Mbit/s ring contains several small slots around it, each slot consisting of 16 bits of data, an 8-bit source address, an 8-bit destination address, a bit telling whether the slot is full or empty, and a few other control bits. To transmit, a host interface just waits for a free slot and fills it up. When the slot arrives at the destination, the receiving interface sets the control bits telling whether it was accepted or not. About 10 μs after transmission, the slot comes back around again so that the sender can find out what happened to it. The sender is not permitted to reuse the slot immediately, as an antihogging measure. By having such small slots and preventing their immediate reuse, the ring guarantees an extremely short delay for small packets, but at the price of higher overhead than the token ring under heavy load.

Still another design is discussed in LIU78. In Liu's design, each ring interface has a shift register equal in length to the maximum packet size. When a host wants to send a packet, it loads up the shift register and inserts the shift register into the ring between two packets. This mechanism leads to low delay, since a host need only wait until the current packet has passed through. When the shift register becomes empty (through a period of low traffic), it can be removed from the ring.

From the above discussion, it should be obvious that many local network protocols have already been devised, with more being threatened all the time. Without standards, the most likely development would be a proliferation of local networks from various vendors, all incompatible; vendor A's terminal would not talk to vendor B's CPU because they would have different protocols embedded in their hardware. In an attempt to nip this incipient chaos in the bud, the Institute of Electrical and Electronics Engineers (IEEE) set up a committee in February 1980 to develop a standard for local network protocols. Although the standard, IEEE 802, was not completed at the time of this writing, the general picture looks as if it will probably be as follows:

The standard treats the physical and data link layers. The physical layer allows for base-band, broad-band, and fiber optics communication, and describes the interfacing of the host (DTE) to the cable. The data link layer handles channel access, as mentioned earlier, as well as addressing, frame format, and control.

The data link layer is split up into two sublayers, media access and data link control, with a third optional sublayer for internetworking (whose presence in the data link layer instead of in the network layer is certainly arguable). The media access sublayer handles channel allocation. It is here that a choice had to be made between CSMA/CD and some kind of ring. The arguments for CSMA/CD were that it was fair, easy to implement on a single chip, had six years of operational experience, and had three major companies (DEC, Intel, and Xerox) already publicly committed to it.

The token ring supporters' counterarguments were as follows: rings, unlike CSMA/CD, provide a guaranteed worst case access time (needed for real-time work, such as speech transmission); rings can be logical as well as physical, accommodating various topologies and allowing important hosts better access by inserting them into the logical ring in several places; and ring performance does not degrade at high load, as does CSMA/CD owing to the many collisions. Unfortunately, neither camp had the necessary two-thirds majority required by IEEE rules, and so a compromise was made in which both CSMA/CD and a token ring were included.

The data link control sublayer was designed to be as compatible with HDLC as possible, on the theory that the last thing the world needed was yet another brand-new data link protocol. Two types of service are provided for: connection oriented and pure datagram. In the former, the data link layer times out and retransmits lost frames, guarantees arrival in sequence, and regulates flow using the standard HDLC sliding-window protocol. In the latter, the data link layer guarantees nothing; once sent, the frame is forgotten (at least by the data link layer).

The major difference between the 802 frame and HDLC's is the presence in 802 of two addresses, source and destination, instead of the one address in HDLC, and the use of variable-length addresses (from 1 to 7 bytes), instead of fixed-length, 1-byte addresses. HDLC was designed for two-party, point-to-point lines, where no address is needed, and for multipoint master/slave lines, in which only the slave's address is needed. In contrast, 802 is aimed at multipoint symmetric lines, where any machine can send to any other machine, and so two addresses are required. The decision to have variable-length addresses up to 7 bytes is intended to allow processes to be designated by a worldwide unique address. Three of the 7 bytes are to be administered by an as-yet-unidentified international organization, and 4 are for local use. Most networks will only need 1- or 2-byte addresses for internal traffic.

3. THE NETWORK LAYER

When a frame arrives at an IMP in a point-to-point network, the data link layer strips off the data link header and trailer and passes what is left, called a *packet*, to the network layer. The network layer must then decide which outgoing line to forward the packet on. It would be nice if such decisions could be made so as to avoid having some lines congested and others idle. Hence congestion control is intimately related to routing. We first look at routing and then at congestion control, both for point-to-point networks. With the channel acquisition protocol for broadcast networks in the data link layer, the network layer for these networks is essentially empty.

Two opposing philosophies exist concerning the network layer. In most local networks and some long-haul networks, the network layer provides a service for delivering independent packets from source to destination with a high probability of success (although less than 1.0). Each packet carried is unrelated to any other packet, past or future, and must therefore carry a full destination address. Such packets are called *datagrams*.

The other approach, taken in many public data networks (especially in Europe), is to require a transmitter to first send a setup packet. The setup packet chooses a route for subsequent traffic and initializes the IMPs along the route accordingly. The user chooses, or is given, a *virtual circuit number* to use for subsequent packets going to the same destination. In this organization, data packets belonging to a single conversation are not independent, since they all follow the same route, determined by the virtual circuit number in them.

The advantage of using virtual circuits ıs that it guarantees that packets will be delivered in order and helps reduce congestion by making it possible to reserve resources (e.g., buffers) along the route in advance. The disadvantage is that a lot of IMP table space is taken up by idle connections and that there is a lot of overhead in setting up and closing down circuits, the latter a great concern in transaction-oriented database systems [MANN78]. With a datagram system, a query-response requires just two packets. With a virtual circuit system it requires six packets: setup, acknowledgment, query, response, close circuit, and acknowledgment.

3.1 Routing in Point-to-Point Networks

Many routing algorithms have been proposed, for example, BARA64, FRAT73, McQU74, RUDI76, SCHW80, and SEGA81. Below we sketch a few of the more interesting ones. The simplest algorithm is *static* or *directory* routing, in which each IMP has a table indexed by destination, telling which outgoing line to use. When a packet comes in, the destination address is extracted from the network layer header and used as an index into the routing table. The packet is then passed back down to the data link layer (see Figure 2) along with the chosen line number.

A variant algorithm provides two or more outgoing lines for each destination, each with a weight. When a packet arrives, a line is chosen with a probability proportional to its weight. Allowing alternatives eases congestion by spreading the traffic around. Note that when virtual circuits are used *within the subnet*, the routing decision is only made for setup packets, not data packets.

Several proposals have been made for determining the routes to be put in the tables. Shortest path routing, which minimizes the number of hops (IMP-IMP lines), is an obvious candidate. In FRAT73 another method, based on flow deviation, is given.

The problem with static routing is just that—it is static—it does not adapt to changing traffic patterns and does not try to route packets around congested areas. One way to have the network adapt is to have one host function as a routing control center. All IMPs send it periodic reports on their queue lengths and line utilizations, from which it computes the best routes and distributes the new routing tables back to the IMPs.

Although seemingly attractive, centralized routing has more than its share of problems [McQU74]. To start with, if the routing control center malfunctions, the network will probably be in big trouble. Second, the complete optimal routing calculation for a large network may require a large dedicated host and even then may not be able to keep up with the traffic fluctuations. Third, since IMPs near the routing control center get their new tables before more distant IMPs do, the network will operate with mixed old-new tables occasionally, a situation that may cause traffic (including the new routing tables) to loop. Fourth, if the network is large, the traffic flow into and out of the routing control center may itself get to be a problem.

One of the earliest routing algorithms [BARA64] adapts to changing traffic, but does so without any central control. In *hot-potato routing*, when a packet arrives, it is assigned to the output line which has the shortest transmission queue. This strategy gets rid of the packet as fast as possible, without regard to where it is going. A much better idea is to combine static information about the suitability of a given output line with the queue lengths. This variant is known as *shortest queue plus bias*. It could be parameterized, for example, to use the shortest queue unless the line is going the wrong way, or to use the statically best line unless its queue exceeds some threshold.

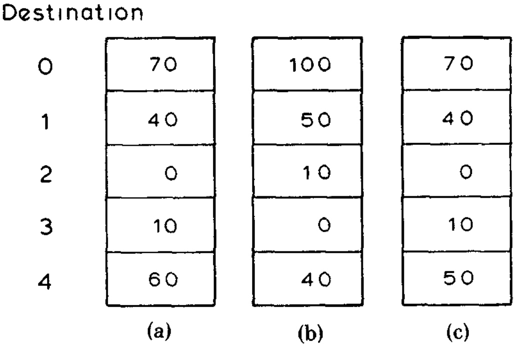
Algorithms like this are known as *isolated adaptive* algorithms [McQu74].

Rudin's *delta routing* [Rudi76] combines some features from centralized and isolated adaptive algorithms. In this method, IMPs send periodic status reports to the routing control center, which then computes the k best paths from each source to each destination. It considers the top few paths equivalent if they differ (in length, estimated transit time, etc.) by an amount less than some parameter δ. Each IMP is given the list of equivalent paths for each destination, from which it may make a choice based on local factors such as queue lengths. If δ is small, only the best path is given to the IMPs, resulting in centralized routing. If δ is large, all paths are considered equivalent, producing isolated adaptive routing. Intermediate strategies are obviously also possible.

A completely different approach is distributed adaptive routing [McQu74], first used in the ARPANET, but replaced after ten years owing to the problems with looping discussed below. With this algorithm, each IMP maintains a table indexed by destination giving the estimated time to get to each destination and also which line to use. The IMP also maintains an estimate of how long a newly arrived packet would take to reach each neighbor, which depends on the queue length for the line to that neighbor.

Periodically, each IMP sends its routing table to each neighbor. When a routing table comes in, the IMP performs the following calculation for each destination. If the time to get the neighbor plus the neighbor's estimate of the time to get to the destination is less than the IMP's current estimate to that destination, packets to that destination should henceforth be routed to the neighbor.

As a simple example, consider a five-IMP network. At a certain instant, IMP 2 has estimates to all possible destinations, as shown in Figure 8a. Suddenly the routing table from IMP 3 (assumed to be adjacent to IMP 2) arrives, as shown in Figure 8b. Let us assume that IMP 2 estimates the delay to IMP 3 to be 10 ms, on the basis of the size of its transmission queue for IMP 3. IMP 2 now calculates that the transit time to IMP 0 via IMP 3 is 10 + 100 ms. Since this time is worse than the 70 ms for its current route, no change is made to entry 0 of the table. Similarly 10 + 50 > 40, and so no change is made for destination 1 either. However, for destination 4, IMP 3 offers a 40-ms delay, which, when combined with the 10-ms delay to get to IMP 3, is still better than the current route (10 + 40 < 60). Therefore, IMP 2 changes its estimate of the time required to get the IMP 4 to 50 ms, and records the line to IMP 3 as the way to get there. The new routing table is given in Figure 8c.



Destination	(a)	(b)	(c)
0	70	100	70
1	40	50	40
2	0	10	0
3	10	0	10
4	60	40	50

Figure 8. Distributed adaptive routing. (a) An IMP's original routing table. (b) Routing table arriving from a neighboring IMP (c) The new routing table, assuming a 10-ms delay to the neighbor.

Although this method seems simple and elegant, it has a problem. Suppose that *A*, *B*, and *C* are connected by lines *AB* and *BC*. If number of hops is used as a metric, *B* thinks it is one hop from *A*, and *C* thinks it is two hops from *A*. Now imagine that line *AB* goes down. *B* detects the dead line directly and realizes that its delay to *A* is now infinite via *AB*. Sooner or later, however, *C* offers *B* a route to *A* of length two hops. *B*, knowing that line *AB* is useless, accepts the offer, and modifies its tables to show that *A* is three hops away via *C*. At this point *B* is routing packets destined for *A* to *C*, and *C* is sending them right back again. Having packets loop forever is not considered a good property to have in one's routing algorithm. This particular problem causes great anguish for the transport layer, as we shall see shortly.

To get around the problem of looping packets, several researchers (e.g., Chu78, Sega81) have proposed using the *optimal-*

ity principle to guarantee loop-free routing. This principle states that if B is on the optimal route from A to C, then the best route from B to C falls along the same route. Clearly if there were a better route from B to C, the best route from A to C could use it too. Consequently, the set of best routes to C (or any other destination) from all other IMPs forms a tree rooted at C. By explicitly maintaining all the trees, the routing algorithm can adapt but prevent looping. A good survey of routing algorithms can be found in SCHW80.

3.2 Congestion Control in Point-to-Point Networks

Now we turn to the problem of congestion in point-to-point networks. Actually, little is known about how to deal with it, and all the proposed solutions are rather ad hoc. Davies [DAVI72] suggested starting each network with a collection of special packets called *permits* that would roam about randomly. Whenever a host wanted to send a packet, its IMP would have to capture and destroy a permit before the new packet could be injected into the network.

This mechanism guarantees that the maximum number of packets in the network can never exceed the initial number of permits, which helps somewhat, but still does not guarantee that all the legal packets will not someday end up in one IMP, overloading it. Furthermore, no one has been able to devise a way to regenerate permits lost in IMP crashes (short of deadstarting the whole network, which is unacceptable). If these permits are not generated, carrying capacity will be permanently lost.

Another congestion control scheme is due to Irland [IRLA78]. This scheme calls for IMPs to monitor the utilization of each outgoing line. When a line utilization moves above a trigger value, the IMP sends a *choke packet* back to the source of each new packet needing that line, telling the source to slow down.

Kamoun [KAMO81] has proposed a congestion control scheme based on the observation that when packets must be discarded in an overloaded IMP, some packets are better candidates than others. In particular, if a packet has already made k hops, throwing it away amounts to discarding the investment in resources required to make those k hops. This observation suggests discarding packets with the smallest k values first. A variation of this idea that does not require a hop counter in each packet is to have IMPs discard newly injected packets from local hosts in order to salvage transit traffic with $k \geq 1$.

The limiting case of a congested network is a deadlocked network. If hosts A, B, and C are all full (no free buffers), and A is trying to send to B and B is trying to send to C and C is trying to send to A, a deadlock can occur, as shown in Figure 9.

Merlin and Schweitzer [MERL80a, MERL80b] describe several ways to prevent this kind of *store-and-forward deadlock* from occurring. One way is to provide each IMP with $m + 1$ packet buffers, where m is the longest path in the network. A packet newly arriving in an IMP from a local host goes into buffer 0. At the next IMP along the path it goes in buffer 1. At the following IMP it uses buffer 2. After having made k hops, it goes in buffer k. To see that the algorithm is deadlock free, consider the set of all buffers labeled m. Each buffer is in one of three states:

(1) empty,
(2) holding a packet destined for a local host,
(3) holding a packet destined for a distant host.

In Case 2 the packet can be delivered and the buffer freed. In Case 3 the packet is looping and must be discarded. In all cases the complete set of buffers labeled m can be made empty. Consequently, all packets in buffers labeled $m - 1$ can be either delivered or discarded, one at a time. The process can then be repeated, freeing the buffers labeled $m - 2$, and so on.

Other kinds of deadlocks in computer networks are discussed in GUNT81.

3.3 An Example Network Layer Protocol: X.25

To help standardize public long-haul networks, CCITT has devised a three-layer protocol of its own. The physical layer is X.21 (or X.21 bis, a stopgap analog interface to be used until the digital network arrives).

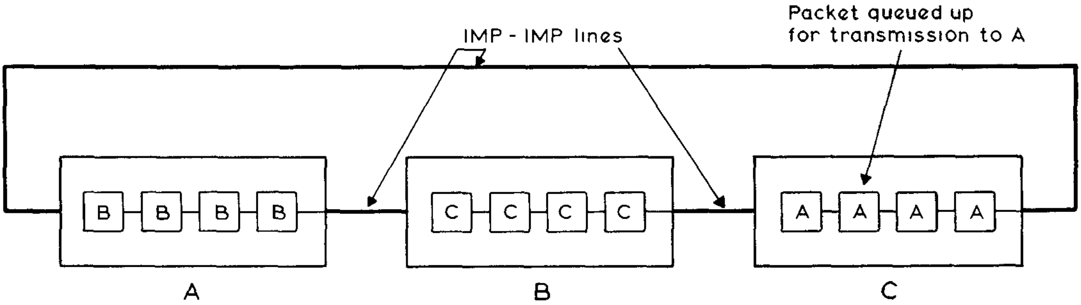


Figure 9. Store-and-forward lockup (deadlock).

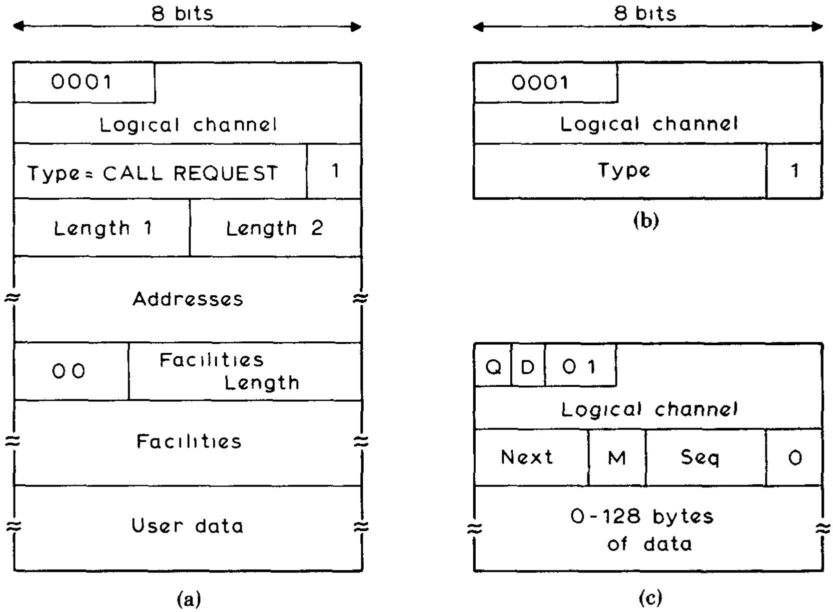


Figure 10. X.25 headers. (a) CALL REQUEST packet (b) Control packet. (c) Data packet.

The data link layer consists of two variants of HDLC (LAP and LAPB). Whether the next layer is network layer protocol or a transport layer protocol is a matter of some debate in the network community. Let us call it a network layer protocol and discuss it now.

X.25 (which is the collective name for all three layers) is virtual circuit oriented [RYBC80]. To set up a virtual circuit, a host (DTE) sends a CALL REQUEST packet into the network. The remote host can either accept or reject the incoming call. If it accepts it, the virtual circuit is set up; otherwise the circuit is cleared.

Figure 10a shows the format of the CALL REQUEST packet. The first 4 bits are 0001. The next 12 bits are the virtual circuit number chosen by the originating host. The third byte is the type code of CALL REQUEST. The next byte gives the number of decimal digits in the caller's and callee's addresses, followed by up to 30 bytes containing the addresses themselves in binary coded decimal. (The telephone community has been using decimal numbers for 100 years, and old habits die hard.) The *Facilities* field is used to request services such as calling collect. Since the facilities field is variable length, a length field is needed. Finally, the user data field can be used in any way the user chooses, for example, to indicate which process within the called host expects the call.

When the CALL REQUEST packet arrives at the destination, that machine accepts or rejects the call by sending a packet of the form shown in Figure 10b. Acceptance or rejection is indicated in the *Type* field. Once the virtual circuit has been set up, both sides may send data packets at will, which makes the connection, by definition, full duplex. Either side may terminate the call by sending a CLEAR REQUEST packet, which is acknowledged by a CLEAR CONFIRMATION packet.

An ordinary data packet is shown in Figure 10c. The *Sequence* and *Next* fields are analogous to those in HDLC. Like HDLC, X.25 layer 3 also has an optional format with 7-bit sequence numbers. The *M* bit can be used by a host to indicate that more data follow in the next packet, thus partitioning the packet stream into multipacket units.

The meaning of the *Q* bit is not specified, but it is provided to allow the transport layer a means for distinguishing transport layer data packets from control packets. The *D* bit stands for Delivery confirmation. If a host sets it on all the packets sent on a certain virtual circuit, the *Next* field will contain a true acknowledgment from the remote host, producing an end-to-end confirmation. If, however, it is always set to 0, then the *Next* field just means that the local IMP (DCE) received the packet specified, not that the remote host did. Conceivably, when $D = 0$, the local IMP could write all the packets on magnetic tape to be mailed to the remote IMP for delivery in a couple of days (bargain basement service).

In the original version of X.25, only $D = 0$ was provided. That point generated so much controversy that delivery confirmation was added later, as was a pure datagram facility and something called *Fast Select.* With the Fast Select facility, the user data field in the CALL REQUEST packet is extended to 128 bytes and a similar field is added to the CLEAR REQUEST packet (used to reject incoming calls). Thus a host can send a short query in the CALL REQUEST packet and get the reply in the CLEAR REQUEST packet, without having to open a virtual circuit.

Because layers 2 and 3 in X.25 have so much overlap, it is perhaps useful to point out that the layer 2 sequence numbers and acknowledgments refer to the traffic between host and IMP for all virtual circuits combined. If a host sends the IMP seven packets (frames), each one for a different virtual circuit, the host must stop sending until an acknowledgment comes back. The layer 2 protocol is required to keep the host from flooding the IMP. In contrast, in layer 3, the sequence numbers are per virtual circuit and therefore flow control each connection separately.

X.25 layer 3 also has a few control packets. These include RESET and RESET CONFIRMATION, used to reset a virtual circuit; RESTART and RESTART CONFIRMATION, used to reset all virtual circuits after a host or IMP crash; RECEIVER READY, used for acknowledgments; RECEIVER NOT READY, used to indicate temporary problems and stop the other side even though the window is not full; and INTERRUPT and INTERRUPT CONFIRMATION, used to send out-of-band signals, such as breaks. All these control packets use the format of Figure 10b, in some cases augmented with an additional byte or two for additional information.

4. THE TRANSPORT LAYER

The network layer does not necessarily ensure that the bit stream sent by the source arrives intact at the destination. Packets may be lost or reordered, for example, owing to malfunctioning IMP hardware or software. The X.25 standard provides a mechanism (RESET and RESTART packets) for the network to announce to a host that it has crashed and lost track of both the current sequence numbers and any packets that may have been in transit. To provide truly reliable end-to-end (i.e., host-to-host) communication, another layer of protocol is needed: the transport layer. (Note that X.25 with $D = 1$ comes close to being end to end, but is not quite enough since it provides no way to transparently recover from network RESETs and RESTARTs.)

Another way of looking at the transport layer is to say that its task is to provide a network independent *transport service* to the session layer. The session layer should not have to worry about any of the implementation details of the actual network. They must all be hidden by the transport layer, analogous to the way a compiler must hide the actual machine instructions from the user of a problem-oriented programming language.

4.1 The Transport Station

The program within the host that implements the transport service is called the *transport station*. Its chief functions are to manage connection establishment and teardown, flow control, buffering, and multiplexing. Although a transport station might conceivably offer only datagram primitives to its users, most offer (and emphasize) virtual-circuit primitives. As a bare minimum, the following primitives or their equivalents are normally available:

```
connum = CONNECT(local, remote),
connum = LISTEN(local),
  status = CLOSE(connum),
  status = SEND(connum, buffer, bytes),
  status = RECEIVE(connum, buffer,
           bytes)
```

The primitives for establishing a transport connection, CONNECT and LISTEN, take *transport addresses* as parameters. Each transport address uniquely identifies a specific transport station and a specific *port* (connection endpoint) within that transport station. For example, CCITT has decreed that X.25 will use 14-digit numbers for addressing. The first three identify the country, and the fourth identifies the network within the country. (Multiple country codes have been assigned to countries that expect to have more than ten public networks.) The last ten digits of the X.25 address are assigned by each network operator, for example, five digits to indicate hosts and five digits for the hosts to allocate themselves.

In our example, the LISTEN command tells the transport station that the process executing it is prepared to accept connections addressed to the indicated local address. The process executing the LISTEN is blocked until the connection is established, at which time it is released, with the variable *connum* being set to indicate the *connection number*. The *connection number* is needed because multiple connections may be open at the same time and a subsequent SEND, RECEIVE, or CLOSE must be able to tell which connection is meant. If something goes wrong, an error number can be returned in *connum* (e.g., positive for connection established, negative for error).

The CONNECT command tells the transport station to send a message (e.g., X.25 CALL REQUEST) to another host to establish a connection. When the connection has been established (or rejected, for example, due to illegal addresses), the connection number or error code is returned in *connum*.

An important design issue is what should the transport station do if a CALL REQUEST packet comes in specifying a transport address for which no LISTEN is pending? Should it reject the request immediately, or should it queue the request in the hope that a LISTEN will be done shortly? If the request is queued, should it time out and be purged if no LISTEN is forthcoming within a reasonable time? If so, what happens if the LISTEN finally occurs after the timeout?

In the above example, both LISTEN and CONNECT are blocking primitives, that is, the caller is halted until the command completes. Some transport stations use nonblocking primitives. In other words, both calls complete immediately, perhaps only checking the syntactic validity of the addresses provided. When the connection is finally established, or definitively rejected, the respective processes are interrupted. Some transport stations that use nonblocking primitives also provide a way for a process to cancel an outstanding LISTEN or CONNECT, as well as a method for a listening process to inspect an incoming connection request before deciding to accept or reject it.

The primitive CLOSE speaks for itself. The status returned would normally be "OK" if the connection actually existed and "error" if it did not.

The SEND and RECEIVE primitives do the real work of message passing. For the sake of clarity, we refer to the entities exchanged here as "messages," to distinguish them from the "packets" of the network layer and "frames" of the data link layer. A message will be encased in a packet, which will be inserted into a frame before transmission, of course. SEND specifies the connection on which to send, the buffer address, and the number of bytes. RECEIVE has the same parameters, although here *bytes* might initially contain the buffer size and later be filled in with the size of the received message. Again, both of these could be provided in nonblocking as well as blocking versions.

A more elaborate transport station could offer commands to send and receive datagrams, to send and receive interrupt signals, to reset the connection in the event of error, and to interrogate the status of the other side, a facility particularly useful for recovering from network layer failures.

4.2 Establishing and Closing Connections

As we pointed out earlier, one consequence of adaptive routing is that packets can loop for an indefinite period of time. If a packet gets trapped, the sending transport station will eventually time out and send a duplicate. If the duplicate gets through properly, but the original packet remains trapped for a while, problems can arise when it finally escapes and is delivered. Imagine, for example, what would happen if a message instructing a bank to transfer a large sum of money were stored and later repeated, long after the transaction had already been completed.

A useful first step is to limit the amount of time that a packet can exist in the network. For example, a counter could be put in the packet header. Each time the packet was forwarded, the counter could be decremented. When the counter reached zero, the packet would be discarded. Alternatively, a timestamp in the packet could be used to render it obsolete after a certain interval.

The next step is to have the transport stations use a sequence space so large (e.g., 32 bits) that no packet can live for a complete cycle. As a result, delayed duplicates can always be detected by their sequence numbers. However, if all new connections always start with sequence number 0, packets from previous connections may come back to plague later ones. Therefore, it is necessary to have each new connection initialize its sequence numbers to a value known to be higher than that of any existing packet.

Unfortunately, not even these measures are enough. Since each host has a different range of sequence numbers outstanding, each one must specify the initial sequence number for packets it will send. Assume that sequence numbers are chosen during the call establishment phase. With some bad luck, the following scenario could occur at an instant when *A* wanted to set up a connection with sequence number 100 to *B*:

(1) *A* sends a CALL REQUEST packet with sequence number 100.
(2) The packet is lost.
(3) An old CALL REQUEST with sequence number 50 suddenly arrives at *B*.
(4) *B*'s CALL ACCEPT packet, with sequence number 700, is lost.
(5) An old CALL ACCEPT from *B* with sequence number 650 suddenly arrives at *A*.

At this point the connection is fully established, with *A* about to send packet 100, but *B* expecting packet 50. Similarly, *B* intends to send packet 700, but *A* expects 650. The result is a deadlock.

Tomlinson [TOML75] proposed a connection establishment protocol that works even in the face of delayed control packets. It is called the *three-way handshake*. An example follows (S means sequence, A means acknowledgment):

(1) *A* sends a CALL REQUEST packet with S = 100.
(2) *B* sends a CALL ACCEPTED packet with S = 700, A = 100.
(3) *A* sends a packet with S = 101, A = 700.

Now consider what happens in the face of the same lost and duplicate packets that lead to deadlock above. When *B* receives the CALL REQUEST with S = 50, it replies with S = 700, A = 50. If this packet gets through, *A* sees the bad acknowledgment and rejects the connection. The only way *A* can be spoofed is for an old CALL AC-

CEPTED packet with A = 100 to appear out of the blue. Such a packet could only be generated in response to an *old* CALL REQUEST packet with S = 100, something *A* has not sent for a long time. Sunshine and Dalal (SUNS78] discuss this problem in more detail.

By now you should be convinced that the protocol required to establish a transport layer connection in the face of an unreliable network layer is nontrivial. What about closing a connection? Surely that, at least, is easy: *A* sends *B* a request to close, and *B* sends back a close acknowledgment. Unfortunately, things are not that simple. As an example, let us briefly consider the two-army problem.

Two divisions of the white army are encamped on the opposite walls of a valley occupied by the blue army. If both divisions attack simultaneously, the white army will win; if either division attacks alone, it will be massacred. The white army divisions must synchronize their attack using an unreliable channel (e.g., a messenger subject to capture). Suppose that white's *A* division sends the message: "Let's attack at teatime," and gets the reply "OK." Division *B* has no way of being sure that the reply got back. If it just goes ahead and attacks, it might get slaughtered. Furthermore, *A* is well aware of this line of reasoning and hence may be afraid to attack, even after having received an acknowledgment.

At this point you may be thinking: "Why not use a three-way handshake here?" Unfortunately, it does not work. *A* could confirm receipt of *B*'s acknowledgment, but because this confirmation could get lost, *A* does not know which situation holds:

(1) *B* got the confirmation and the war is on.
(2) The confirmation got lost and the war is off.

How about a four-way handshake? This is no better. An *n*-way handshake? Still no good. In all cases, the sender of the last message cannot tell whether or not it arrived. If its arrival is essential to starting the war, the sender has no way of telling whether the receiver is going to attack or not. If its arrival is not essential to starting the war, one can devise an equivalent protocol not containing it and apply the above reasoning to the new protocol.

The implication of all this is that a closing protocol in which neither side can hang up until it is convinced that the other side also intends to hand up is, at the very least, more complicated than it at first appears. The issue is discussed further in SUNS78 and YEMI79.

4.3 Flow Control and Buffering

An important design issue in the transport layer is flow control. Since no transport station has an infinite amount of buffer space, some way must be provided to prevent a fast sender from inundating a slow receiver. Flow control is well known in other contexts, such as operating systems design, where it is known as the producer-consumer problem. Although it also occurs in the data link and network layers, some new complications are present in the transport layer.

To start with, the data link layer usually has one connection for each adjacent IMP—a handful at most—whereas the transport layer in a large multiprogrammed computer may have many connections open simultaneously. A stop-and-wait protocol in the transport layer is usually undesirable, since both sender and receiver would have to be scheduled and run for every message sent. If each machine had a response time of 500 ms between the moment a process became ready to run and the time it ran, the transport connection could support two messages per second, at best, and probably fewer. Consequently, large windows are needed to achieve high throughput, but the combination of large windows and many open connections necessitates many buffers, most of which are idle most of the time.

An alternative design is not to dedicate buffers to specific connections, but to maintain a buffer pool and pull buffers out of the pool and assign them to connections dynamically, as needed. This strategy entails some risk, since no buffer may be available when one is needed. To avoid this risk, a buffer reservation protocol is needed, increasing traffic and overhead. Furthermore, if the buffers are of fixed size and messages

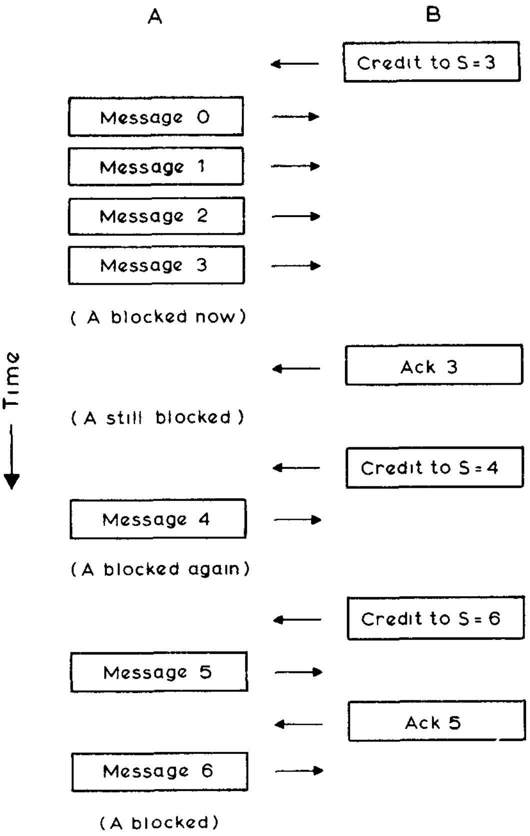


Figure 11. Flow control using credits.

vary from a few characters to thousands of characters, a pool of fixed-sized buffers is not attractive either. No one solution is best. Each transport station must make compromises appropriate to its expected work load.

Another important issue is the relation of flow control to error control. With the sliding-window protocol, an acknowledgment message has two distinct functions: to announce that a message has arrived and to grant the sender permission to send another message. In the transport layer, this coupling is not always desirable.

To see why, consider the dilemma of a transport station that is chronically short of buffer space. What should it do if a message arrives, but the process using the connection has no RECEIVEs outstanding? If it does nothing, the sending transport station will eventually time out and send it again. If it sends an acknowledgment, the other transport station may send yet another message. The problem comes from the fact that the transport station has no control over the rate at which the user does RECEIVEs. Earlier, we more or less assumed that the network layer was always hungry for new packets—a reasonable assumption, since the network code has little else to do.

One way out of this dilemma is to decouple acknowledgments and flow control. To do so, we introduce two kinds of control messages: acknowledgments and credits. An acknowledgment simply says that a certain message (and by implication, all lower numbered messages) has arrived safely. Upon receiving an acknowledgment, the sender may release the buffers containing all the acknowledged messages, since none of them will ever be retransmitted. However, an acknowledgment does *not* imply permission to send any more messages.

Such permission is granted by a credit message. When a connection is established, the receiver grants some credits to the sender. These credits may be for so many messages or so many bits or both. Every time a message is sent, the credits for message count and/or bit count are decremented. When the credits are all used up, the sender must stop sending until more credits arrive. Such credits may be sent as distinct messages or they may be piggybacked onto data or acknowledgment messages. This scheme provides a simple and flexible mechanism for preventing unnecessary retransmissions in the presence of heavy and variable demands on limited buffer space. An example is given in Figure 11.

4.4 Connection Multiplexing

Multiplexing of connections plays an important role in several layers. In lower layers, for example, packets and frames ultimately destined for different hosts are multiplexed onto the same output lines. In the transport layer, two different forms of multiplexing occur. In *upward multiplexing* (shown in Figure 12a) several transport connections are multiplexed onto the same network connection (e.g., the X.25 virtual circuit). Upward multiplexing is often financially better, since some carriers charge by the packet and also by the second for each virtual circuit that is open.

Now consider the plight of an organization (e.g., an airline) that has 100 telephone operators to handle customer inquiries. If each operator is assigned to a separate virtual circuit, 100 virtual circuits to the central computer will be open all day. The other option would be to use a single virtual circuit to the computer, with the first byte of data being used to distinguish among the operators. The latter has the disadvantage that if the traffic is heavy, the flow control window may always be full, thus slowing down operation. With a dedicated virtual circuit per operator, full windows are much less likely to occur.

The other form of multiplexing, *downward multiplexing* (Figure 12b), becomes interesting when the network layer window is too small. Suppose, for example, a certain network offers X.25, but does not support the 7-bit sequence number option. A user with a large number of data to send might find himself constantly running up against full windows. One way to make an end run around the problem is to open multiple virtual circuits for a single-transport connection. Packets could be distributed among the virtual circuits in a round-robin fashion, first a packet on circuit 0, then a packet on circuit 1, then one on circuit 2, and so on.

Conceivably the two forms could even be combined. For *n* connections, *k* virtual circuits could be set up with traffic being dynamically assigned.

5. THE SESSION LAYER

In many networks, the transport layer establishes and maintains connections between hosts. The session layer establishes and maintains connections, called *sessions*, between specific pairs of processes. On the other hand, some networks ignore the session layer altogether and maintain transport connections between specific processes. The ISO OSI Reference Model is exasperatingly vague on this point, stating only that the session layer connects "presentation-entities" and that the transport layer connects "session-entities."

To keep the following discussion from vanishing in a linguistic fog, we assume that transport connections are between hosts and session connections are between processes. Thus, when a process wants to talk to another process, it makes its desires known to the session layer, which then engages the services of the transport layer to set up a transport connection to the remote host for use by the session.

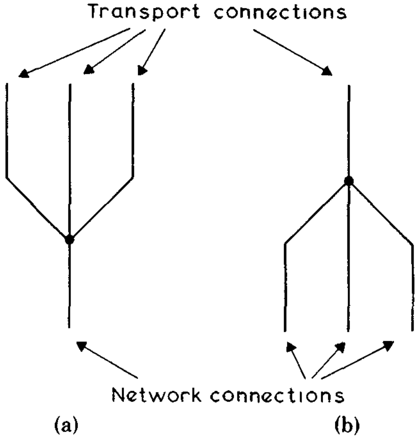


Figure 12. (a) Upward multiplexing. (b) Downward multiplexing.

A principal task of the session layer is to connect two processes together into a session. Since it is inconvenient for users to be aware of hard transport addresses, the session layer could allow them to refer to destinations by symbolic name, with the session layer doing the mapping onto transport addresses. For example, a user could say, in effect, "Give me a phototypesetter process," with the session layer worrying about where such beasts were to be found.

When a session is set up, an activity often call *session binding*, certain conventions about the coming session can be established. Typical conventions are half-duplex versus full-duplex data transfer, character codes, flow control window sizes, the presence or absence of encryption or text compression, and how to recover from transport layer failures.

Another task that the session layer can perform is particularly useful in networks where the user primitives for sending and receiving messages are nonblocking, and where the user may have multiple requests outstanding on the same session at any instant. Under these circumstances, replies may come back in an order different from that in which the requests were sent. The session layer's *dialog control* function can keep track of requests and replies and reorder them if need be to simplify the design of the user programs.

Another aspect of dialog control is bracketing groups of messages into atomic units. In many database applications it is highly undesirable that a transaction be broken off part way, as a result of a network failure, for example. If the transactions consists of a group of messages, the session layer could make sure that the entire group had been successfully received at the destination before even attempting to start the transaction.

Our discussion of the session layer is now complete. The brevity of this section is directly related to the fact that few networks make much of a distinction between the transport and session layers. In fact, many networks have neither a session layer nor any of the dialog control functions belonging to the session layer. While there are no internationally accepted standards for the transport layer yet, there are at least a few serious proposals that have been under discussion for several years [DEPA76, INWG78]. Session layer protocols have not come as far yet. This situation has occurred because the protocol community has been tackling the layers more or less bottom up and is currently in the vicinity of layer 4. Higher layer standards will no doubt be forthcoming in the future.

6. THE PRESENTATION LAYER

The function of the presentation layer is to perform certain generally useful transformations on the data before they are sent to the session layer. Typical transformations are text compression, encryption, and conversion to and from network standards for terminals and files. We examine each of these subjects in turn.

6.1 Text Compression

Bandwidth is money. Sending thousands of trailing blanks across a network to be "printed" is a good use of neither. Although the network designers could leave the matter of text compression to each user program, it is more efficient and convenient to put it into the network architecture as one of the standard presentation services.

Obvious candidates for text compression are runs of repeated bits (e.g., leading zeros) and repeated characters (e.g., trailing blanks). Huffman coding is also a possibility. Since text compression is such a well-known subject outside the network context (see, e.g., DAVS76), we do not consider it further here.

6.2 Encryption Protocols

Information often has great economic value. As an example, just think about the data transmitted back by oil companies from exploratory sites. With more and more data being transmitted by satellite, the problem of data security looms ever larger. The financial incentive to erect an antenna to spy on competitors is great and the cost is low. Furthermore, privacy legislation in many countries puts a legal requirement on the owners of personal data to make sure such data are kept secret. All these factors combine to make data encryption an essential part of most networks. The December 1979 issue of *Computing Surveys* [COMP79] is devoted to cryptography and contains several introductory articles on it.

An interesting question is: "In which layer does the encryption belong?" In our view, encryption is analogous to text compression: ordinary data go in and compressed or indecipherable data come out. Since everyone agrees that text compression is a presentation service, logically encryption should be too. For historical reasons and implementation convenience, however, it is often put elsewhere, typically the transport layer or the data link layer.

The purpose of encryption is to transform the input, or *plaintext*, into an output, or *ciphertext*, that is incomprehensible to anyone not privy to the secret *key* used to parameterize the transformation. Thus plaintext is converted to ciphertext in the presentation layer of the source machine and reconverted to plaintext in the presentation layer of the destination machine. In neither machine should the user programs be aware of the encryption, other than having specified encryption as an option when the session was bound.

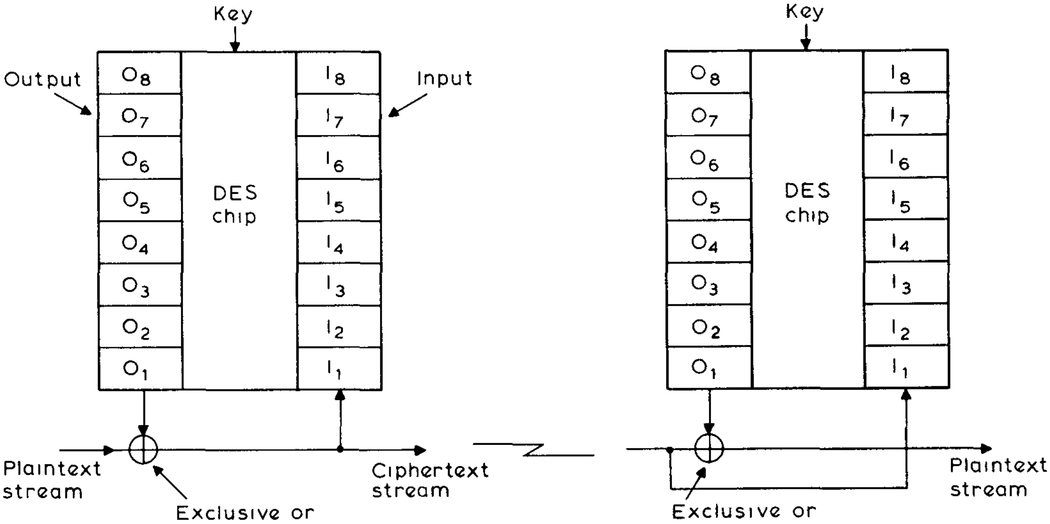


Figure 13. A stream cipher using DES. Data arrives from the left and is encrypted for transmission. The destination machine decrypts it and outputs the plaintext.

One of the best-known encryption methods is the *substitution cipher*, in which a unit of plaintext is converted into a unit of ciphertext. In a *monoalphabetic cipher*, each letter is converted into another letter according to a fixed rule. For example, "a" becomes "M," "b" becomes "R," "c" becomes "G," etc. In this example, the encryption key is MRG ..., that is, the ciphertext corresponding to the plaintext abc.... Although 26! different monoalphabetic substitutions exist, these ciphers can be broken by a clever ten-year-old using the frequency statistics of natural language.

Most computer ciphers use the same principle, but on a larger scale. The U.S. federal government has adopted a substitution cipher that is fast becoming a de facto standard for nongovernmental organizations as well. The *DES* (Data Encryption Standard) cipher takes a 64-bit plaintext input block and produces a 64-bit ciphertext output block. The transformation is driven by a 56-bit key. Conceptually, at least, one could prepare a big table, with 2^{64} columns, one for each possible input, and 2^{56} rows, one for each possible key. Each table entry is the ciphertext for the specified input and key.

DES can also be operated as a *stream cipher*, as shown in Figure 13. The input shift registers on both source and destination machine are initialized to the same 8-byte (random) number, $I_1, \ldots, I_8$. Data are presented for encryption 1 byte at a time, not 8 bytes at a time. When a byte arrives, the DES chip converts the 8 bytes $I_1, \ldots, I_8$ into the output $O_1, \ldots, O_8$. Then O_1 is Exclusive Or'ed with the input to form the ciphertext byte. The ciphertext byte is both transmitted and fed back into I_1, shifting I_2 to I_3, and so on. I_8 is shifted out and lost. Decryption at the other end is similar. Note that feeding back the ciphertext into the DES input register makes subsequent encryption dependent on the entire previous plaintext, and so a given sequence of 8 plaintext bytes will have a different ciphertext on each appearance in the plaintext.

DES has been the subject of great controversy since its inception [DAVA79, DIFF76a, HELL79, HELL80]. Some computer scientists feel that a wealthy and determined intruder who knew, for example, that a certain message was in ASCII, could determine the key by trying all keys until he found one that yielded ASCII plaintext (i.e., only codes 0–127 and not 128–255). If the ciphertext is k bytes long, the probability of an incorrect key yielding ASCII input is 2^{-k}. For even a single line of text, it is unlikely that any key but the correct one could pass the test.

The dispute centers about how much a DES-breaking machine would cost. In 1977, Diffie and Hellman [DIFF77] designed one and computed its cost at 20 million dollars.

The DES supporters say this figure is too low by a factor of 10, although even they concede DES cannot hold out forever against the exponential growth of very large-scale integrated circuits.

To use DES, both the source and destination must use the same key. Obviously the session key cannot be sent through the network in plaintext form. Instead, a master key is hand carried in a locked briefcase to each host. When a session is set up, a key manager process somewhere in the network picks a random key as session key, encrypts it using the master key, and sends it to both parties for decryption. Since the plaintext of this message is a random number, it is hard to break the cipher using statistical techniques. Numerous variations of the idea exist, typically with master keys, regional keys, local keys, and the like.

Shamir [SHAM79] has devised a clever way to share (master) keys in a flexible way among a large group of people, so that n arbitrary people can get together and assemble the master key, but $n - 1$ people can gain no information at all. Basically, each person is given a data point that lies on a degree $n - 1$ polynomial whose y intercept is the key. With n data points, the polynomial, hence the key, is uniquely determined, but with $n - 1$ data points it is not. Modulo arithmetic is used for obfuscatory purposes.

All the master key methods have a significant drawback, though: it is impossible for computers that have not previously had any contact with each other to agree on a session key in a secure way. Considerable academic research has been done on this topic in recent years (not without its own controversy—see SHAP77 and SUGA79), and some interesting results have been achieved. Merkle [MERK78a], for example, has suggested that two strangers, A and B, could establish a key as follows. A sends k ciphertext messages to B with the instruction to pick one of them at random and break it by brute force (i.e., try all possible keys until a plaintext starting with 64 0's appeared). The rest of the message consists of two random numbers, the key number and the key itself. Having broken the cipher, B then sends the key number back to A to indicate which message was broken. Clearly an intruder will have to break $k/2$ messages on the average to find the right one. By adjusting k and the difficulty of breaking a message, A can achieve any degree of security desired.

A completely different approach to key distribution is that of *public key cryptography* [DIFF76b] in which each network user deposits an encryption key E in a publicly readable file. The user keeps the decryption key D secret. The keys must satisfy the property that $D(E(P)) = P$ for an arbitrary plaintext P. (This is essentially the definition of a decryption key.) The cipher system must be such that D cannot be deduced from the publicly known E.

With this background, the encryption system is obvious and trivial: to send a message to a stranger, you just encrypt it with his publicly known key. Only he knows the decryption key and no one can deduce it from the encryption key, so the cipher cannot be broken. The utility of the whole system depends on the availability of key pairs with the requisite properties. Much effort has gone into searching for ways to produce such key pairs. Some algorithms have already been published [MERK78b, RIVE78, SHAM80]. The scheme of Rivest et al. effectively depends on the fact that given two huge prime numbers, generating their product (the public key) is computationally easy, but given the product, finding the prime factors (the secret key) is very hard. In effect, their system takes advantage of the fact that the computational complexity of factorization is high.

Another area where cryptography plays a major role is in authentication. Suppose that a customer's computer instructs a bank's computer to buy a ton of gold and debit a certain account. The bank complies, but the next day the price of gold drops sharply and the customer denies ever having issued any purchase order. How can the bank protect itself against such unscrupulous customers? Traditionally, court battles over such matters have focused on the presence or absence of an authorized handwritten signature on a piece of paper. With electronic funds transfers and similar applications the need for "digital" (i.e., electronic) signatures is obvious.

With a slight additional restriction, public key cryptography can be used to provide these badly needed digital signatures. The restriction is that the encryption and decryption algorithms be chosen so that $D(E(P)) = E(D(P))$. In other words, the order of applying encryption and decryption must be interchangeable. The M.I.T. algorithm [RIVE78] has this property.

Now let us reconsider the ton-of-gold problem posed earlier. To protect itself, the bank can insist that a customer C use the following protocol for sending signed messages. First, the customer encrypts the plaintext message P with his secret key; that is, the customer computes $D_C(P)$. Then the customer encrypts this result with the bank's public key E_B, yielding $E_B(D_C(P))$. When this message arrives, the bank applies its decryption key D_B to get

$$D_B(E_B(D_C(P))) = D_C(P).$$

Now the bank applies the customer's public key, E_C, to recover P. The bank also saves P and $D_C(P)$ in case trouble arises.

When the angry letter from the customer arrives, the bank takes both P and $D_C(P)$ to court and asks the judge to decrypt the latter using the customer's public key. When the judge sees that the decryption works, he will realize that the bank must be telling the truth. How else could it have come into possession of a message encrypted by D_C, the customer's secret key? Since the bank does not know any of its customer's secret keys, it cannot forge messages (to generate commissions); hence customers are also protected against unscrupulous banks. While in jail, the customer will have ample time to devise interesting new public key algorithms.

Unfortunately, as Saltzer [SALT78] has pointed out, the public key digital signature protocol suffers from some nontechnical problems. For example, immediately after the price of gold drops, the management of the gold-buying company could run to the police claiming it had just become aware of yesterday's key burglary. Depending on local laws, the company might or might not be able to weasel out of obligations undertaken with the "stolen" key. (As an aside, note that the owner of a stolen credit card usually has only a small liability for its subsequent misuse.)

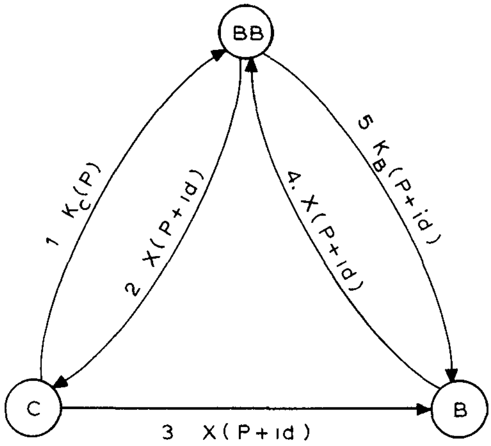


Figure 14. A digital signature protocol with conventional cryptography.

Saltzer also points out that a company is free to change its public key at will. Stronger yet, it may be company policy to do so regularly. If the company changes its key before accusing the bank of fabricating the purchase order, it will be impossible for the bank to convince the judge. This observation suggests that some central key registration authority may be needed. However, if such a central authority, call it Big Brother (BB), exists, conventional cryptography can also be used to achieve digital signatures [NEED78, POPE79].

The signature protocol using DES is illustrated in Figure 14. When a new customer C joins the system, the customer hand carries a secret (DES) key, K_C, to BB. Thus, BB has each user's secret key and can therefore send and receive secure messages from each user. In addition, BB has a secret key of its own, X, that it never discloses to anyone. The protocol for buying gold is as follows (P is the plaintext purchase order):

(1) The customer sends $K_C(P)$ to BB.
(2) BB decrypts the message and returns $X(P$ + identification).
(3) The customer sends $X(P$ + identification) to the bank.
(4) The bank sends $X(P$ + identification) to BB.
(5) BB sends $K_B(P$ + identification) back to the bank.

The "identification" appended to the message by BB consists of the customer's identity, something that BB can guarantee since the incoming message is encrypted by a key only known to one user, plus the date, time, and perhaps a sequence number. Messages encrypted by X can be freely sent through the network, since only BB can decrypt them, and BB is assumed to be trusted. If a dispute arises, the bank can go to the judge with $X(P +$ identification$)$, which the judge can then order BB to decrypt. The judge will then see the identification and know who sent the original message. While in jail, the customer will have ample time to devise interesting new signature protocols using conventional cryptography.

6.3 Virtual-Terminal Protocols

Dozens of brands of terminals are in widespread use, no two of which are identical. Needless to say, a network user who has just been told that the program or host he wishes to use does not converse with his brand of terminal is not likely to be a happy user. For example, if the program treats carriage returns and line feeds as equivalent and the user's terminal only has a "newline" key, which generates one of each, the program will perceive alternate lines as being empty.

To prevent such difficulties, protocols have been invented to try to hide terminal idiosyncracies from application (i.e., user) programs. Such protocols are known as *virtual-terminal protocols*, since they attempt to map real terminals onto a hypothetical network virtual terminal. Virtual-terminal protocols are part of the presentation layer.

Broadly speaking, terminals can be divided up into three classes: scroll mode, page mode, and form mode. Scroll-mode terminals do not have any intelligence. When a key is struck, the character is sent over the line and perhaps printed as well. When a character comes in over the line, it is just displayed. Most hard-copy terminals, and some of the less expensive CRT terminals, are scroll-mode terminals.

Even though scroll-mode terminals are simple, they still can differ in many ways: character set, line length, half duplex/full duplex, overprinting and the way line feed, carriage return, tab, vertical tab, backspace, form feed, and break are handled.

Page-mode terminals are typically CRT terminals with 24 or 25 lines of 80 characters. Most of these have cursor addressing, so that the operator or the program can randomly access the screen. Some of them have a little local editing capability. They have the same potential differences as scroll-mode terminals, and, additionally, problems with screen length, cursor addressing, blinking, reverse video, color, multiple intensities, and the details of the local editing.

Form-mode terminals are sophisticated microprocessor-based devices intended for data entry. They are widely used in airline reservations, banking, and many other applications. In a typical situation the computer displays a form for the operator to fill out using cursor control and local editing facilities. The completed form is then sent back to the computer for processing. Sometimes the microprocessor can perform simple syntax checking, to make sure, for example, that a bank account field contains only numbers.

Two kinds of virtual-terminal protocols are commonly used. The first one is intended for scroll-mode terminals and is based on the ARPANET Telnet protocol [DAVD77]. When this type of protocol is used, the designers invent a fictitious virtual terminal onto which all real terminals are mapped. Application programs output virtual-terminal characters, which are mapped onto the real terminal's character set by the presentation layer at the destination. Supporting a new kind of real terminal thus requires modifying the presentation layer software to effect the new mapping, but does not require changing any of the application programs.

Since most of the current research in virtual-terminal protocols focuses on page- or form-mode terminals, let us move on to them. A general model that has been widely accepted is the data structure model of Schicker and Duenki [SCHI78]. Roughly speaking, protocols based on this approach use the model of Figure 15. Each end of a session has a data structure that represents the state of the virtual terminal.

The data structure consists of a collection

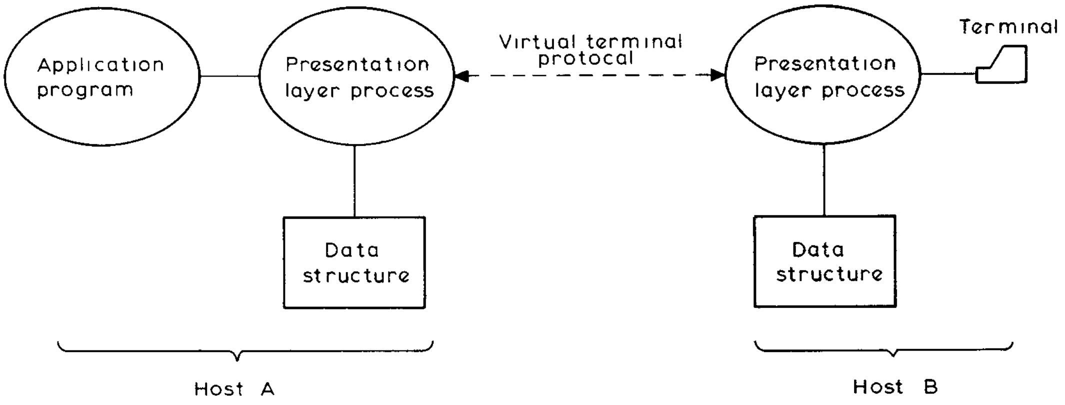

Application program
Presentation layer process
Virtual terminal protocol
Presentation layer process
Terminal
Data structure
Data structure
Host A
Host B

Figure 15. Virtual-terminal protocol model.

of fields, each of which contains certain attributes. Typical attributes are the size of the field, whether it accepts numbers, letters, or both, its rendition (an abstract concept used to model color, reverse video, blinking, and intensity), whether it is protected against operator modification or not, and so forth. The program is written using abstract operations on the data structure. Every time the program changes the data structure on its machine, the presentation layer sends a message to the other machine telling it how to change its data structure. The remote presentation layer is responsible for updating the display on the real terminal to make it correspond to the newly changed data structure. Similarly, changes made to the display by the human operator are reflected in the data structure on the operator's side of the session. Messages are then sent to bring the other side up to date. The protocol used for these messages is the virtual-terminal protocol.

Although a clever presentation layer can come a long way toward hiding the properties of the real terminal from the user program, it cannot work miracles. If the program needs a 24 × 80 screen with cursor addressing and four renditions, the presentation layer will be hard pressed to map everything onto a simple hard-copy terminal. Consequently, virtual-terminal protocols always have an option negotiation facility that is used to establish what each end of the connection is able to provide and what it wants from the other end.

This negotiation can be symmetric or asymmetric. In symmetric negotiation, each end announces its capabilities, inspects its partner's, and sets the parameters to the lowest common denominator. For example, if one end has a 24 × 80 screen, and the other has a 25 × 72 screen, the screen used will be 24 × 72. In asymmetric negotiation, one side makes a proposal and the other side accepts or rejects the proposal. If the proposal is rejected, the proposer may try again. Symmetric negotiation solves the problem of who should go first, but requires more complicated rules to determine what the result of an exchange is. It can also fail, for example, if both sides want to work in alternating (half-duplex) mode, and each wants to go first.

Another important design issue in virtual-terminal protocols is how to handle interrupts (attentions). When a user hits the "break" or "quit" key to terminate an infinite loop with a print statement in it, the presentation layer must purge the pipe of input already queued up; otherwise break will have no apparent effect. It is easy for the presentation layer on the terminal side to begin discarding input upon seeing a break, but it is much harder to determine when to stop discarding. Waiting for the prompt character does not work, since it might occur in the data to be discarded. A special out-of-band signaling protocol is needed. A survey of virtual-terminal protocols is given in DAY80.

6.4 File Transfer Protocols

The most common uses of computer networks at present are for logging onto remote machines and transferring files between machines. These two areas are similar in that just as there is a need for programs to talk to a variety of incompatible terminals, there is a need for programs to read a variety of incompatible files. In principle, the same approach can be used for file transfer as for terminals: define a network standard format and provide a mapping from and to each existing file format.

In practice, this approach seems to work fairly well for terminals, but less well for files, primarily because the differences between terminals are not as great as between file types. Mapping reverse video onto blinking is straightforward compared to mapping 60-bit CDC floating point numbers onto 32-bit IBM floating point numbers, especially when the numbers are strewn randomly throughout the file.

Files are transferred for four primary reasons:

(1) to store a file for subsequent retrieval;
(2) to print a remote file on the local printer;
(3) to submit a file as a remote job;
(4) to use a remote file as data input or output.

Each category of use has its own peculiarities.

When a file is stored for subsequent retrieval, it must be possible to produce an exact, bit-for-bit copy of it upon request. Clearly transmission must be fully transparent, without escape codes that do funny things. The number of bits in the file must be recorded in the stored file, to allow transport between machines with differing word lengths. The last word on the storage machine may be partially full, and so some record of how many bits are in use is required.

When a file is transferred to be printed, problems can arise as a result of different print conventions. Some machines store print files in FORTRAN format, with fixed-length records (with or without some fudge for trailing blanks), and carriage control characters in column 1. Other machines use ASCII style variable-length records, with line feeds and form feeds for indicating vertical motion. When the file is being moved to be used for remote job entry, the same problems are present.

Moving data files containing mixtures of integers, floating point numbers, characters, etc., between machines is nearly impossible. In theory, each data item (e.g., integer, floating point number, character) could occupy one record in a canonical format, with the data type and value both explicitly stored. In practice the idea does not seem to work well, not only because of problems of interfacing existing software to it, but also because of the high overhead and the problems involved in converting floating point numbers from one format to another.

Another aspect of file transfer is file manipulation. Users often need to create, delete, copy, rename, and otherwise manage remote files. Most file transfer protocols tend to concentrate on this aspect of the problem because it is not as hopeless as the conversion aspect. Gien [GIEN78] has described a file transfer protocol in some detail.

7. SUMMARY

Computer networks are designed hierarchically, as a series of independent layers. Processes in a layer correspond with their peers in remote machines using the appropriate protocol, and with their superiors and subordinates in the same machine using the appropriate interface. The ISO OSI Reference Model has been designed to provide a universal framework in which networking can be discussed. Few existing networks follow it closely, but there is a general movement in that direction.

The seven-layer ISO model can be briefly summarized as follows. The physical layer creates a raw bit stream between two machines. The data link layer adds a frame structure to the raw bit stream, and attempts to recover from transmission errors transparently. The network layer handles routing and congestion control. The transport layer provides a network-independent transport service to the session layer. The session layer sets up and manages process-to-process connections. The presentation layer performs a variety of useful conversions. Finally the application layer is up to the user, although some industry-wide protocols may be developed in the future.

The literature on computer networks is huge. Readers unfamiliar with it, but wishing to continue their study of the subject, may be interested in the textbooks by Davies et al. [Davi79] and Tanenbaum [Tane81], or the book edited by Kuo [Kuo81].

ACKNOWLEDGMENTS

I would like to thank Yogen Dalal, Adele Goldberg, and an anonymous, but artistic, technical editor for their numerous and helpful comments.

REFERENCES

Abra70 Abramson, N. "The ALOHA system—another alternative for computer communications," in *Proc. 1970 Fall Jt Computer Conf*, AFIPS Press, Arlington, Va., pp 281-285.

Abra73 Abramson, N "The ALOHA system," in *Computer-communication networks*, N. Abramson and F. Kuo (Eds.), Prentice-Hall, Englewood Cliffs, N.J., 1973.

Bara64 Baran, P. "On distributed communication networks," *IEEE Trans. Commun. Syst.* **CS-12** (March 1964), 1-9.

Bert80 Bertine, H. V. "Physical level protocols," *IEEE Trans. Commun* **COM-28** (April 1980), 433-444.

Bogg80 Boggs, D R., Shoch, J. F., Taft, E. A., and Metcalf, R. M "Pup· An Internet architecture," *IEEE Trans. Commun* **COM-28** (April 1980), 612-624.

Cape79a Capetanakis, J. I "Generalized TDMA: The multi-accessing tree protocol," *IEEE Trans Commun.* **COM-27** (Oct. 1979), 1476-1484.

Cape79b Capetanakis, J. I. "Tree algorithms for packet broadcast channels," *IEEE Trans Inf. Theory* **IT-25** (Sept 1979), 505-515.

Chla76 Chlamtac, I. "Radio packet broadcasted computer network—the broadcast recognition access method," M.S. thesis, Dep Mathematical Sciences, Tel Aviv Univ , Tel Aviv, Israel, 1976.

Chla79 Chlamtac, I., Franta, W. R., and Levin, D. "BRAM: The broadcast recognizing access method," *IEEE Trans Commun.* **COM-27** (Aug. 1979), 1183-1190.

Chu78 Chu, K. "A distributed protocol for updating network topology information," Rep. RC 7235, IBM Thomas J. Watson Res. Cent., Yorktown Heights, N Y., 1978.

Clar78 Clark, D D., Pogran, K. T , and Reed, D P. "An introduction to local area networks," *Proc IEEE* **66** (Nov. 1978), 1497-1517.

Comp79 *Computing Surveys* **11,** 4 (Dec. 1979).

Crow73 Crowther, W., Rettberg, R., Walden, D., Ornstein, S., and Heart, F. "A system for broadcast communication: Reservation-Aloha," in *Proc 6th Hawaii Int. Conf Systems Science,* 1973, pp. 371-374.

Dava79 Davida, G. I. "Hellman's scheme breaks DES in its basic form," *IEEE Spectrum* **16** (July 1979), 39

Davd77 Davidson, J , Hathaway, W., Postel, J., Mimno, N., Thomas, R., and Walden, D. "The ARPANET Telnet protocol: Its purpose, principles, implementation, and impact on host operating system design," in *Proc. 5th Data Communication Symp.* (ACM/IEEE) (1977), pp. 4 10-4.18

Davi72 Davies, D. W. "The control of congestion in packet-switching networks," *IEEE Trans. Commun.* **COM-20** (June 1972), 546-550.

Davi73 Davies, D. W., and Barber, D. L A. *Communication networks for computers*, Wiley, New York, 1973.

Davi79 Davies, D W., Barber, D. L. A., Price, W. L , and Solomonides, C. M. *Computer networks and their protocols,* Wiley, New York, 1979.

Davs76 Davisson, L , and Gray, R. (Eds) *Data compression,* Dowden, Hutchinson & Ross, Stroudsburg, Pa , 1976.

Day80 Day, J. "Terminal protocols," *IEEE Trans. Commun* **COM-28** (April 1980), 585-593.

Depa76 Deparis, M., Duenki, A., Glen, M., Laws, J., LeMoli, G., and Weaving, K. "The implementation of an end-to-end protocol by EIN centers: A survey and comparison," in *Proc. 3rd Int. Conf. Computer Communication* (ICCC) (Aug 1976), pp. 351-360

Diff76a Diffie, W., and Hellman, M E. "A critique of the proposed data encryption standard," *Commun ACM* **19** (March 1976), 164-165.

Diff76b Diffie, W., and Hellman, M. E "New directions in cryptography," *IEEE Trans. Inf Theory* **IT-22** (Nov. 1976), 644-654.

Diff77 Diffie, W., and Hellman, M. E. "Exhaustive cryptanalysis of the NBS data encryption standard," *Computer* **10** (June 1977), 74-84.

Doll78 Doll, D. R. *Data communications.* Wiley, New York, 1978.

Farb72 Farber, D. J., and Larson, K. C "The system architecture of the distributed computer system—the communications system," in *Symp. Computer Networks,* Polytechnic Institute of Brooklyn, Brooklyn, N.Y., April 1972.

Folt80 Folts, H. C. "Procedures for circuit-switched service in synchronous public data networks," *IEEE Trans. Commun* **COM-28** (April 1980), 489-496.

Fras75 Fraser, A. G. "A virtual channel network," *Datamation* **21** (Feb. 1975), 51-56.

Frat73 Fratta, L., Gerla, M., and Kleinrock, L. "The flow deviation method: An approach to store-and-forward communication networks," *Networks* **3** (1973), 97-133.

Free80 Freeman, H. A., and Thurber, K J. "Updated bibliography on local computer networks," *Comput. Arch. News* **8** (April 1980), 20-28.

Gele78 Gelenbe, E., Labetoulle, J., and Pujolle, G. "Performance evaluation of the HDLC protocol," *Comput Networks* **2** (Sept.-Oct 1978), 409-415.

Gerl77 Gerla, M , and Kleinrock, L. "Closed loop stability controls for S-ALOHA satellite communications," in *Proc. 5th Data Communication Symp.* (ACM/IEEE), (1977), pp. 2-10-2-19.

Gien78 Gien, M. "A file transfer protocol (FTP)," *Computer Networks* **2** (Sept.-Oct. 1978), 312-319.

Gunt81 Gunther, K. D. "Prevention of deadlocks in packet-switched data transport systems," *IEEE Trans. Commun.* **COM-29** (April 1981), 512-524.

Hell79 Hellman, M. E. "DES will be totally insecure within ten years," *IEEE Spectrum* **16** (July 1979), 32-39.

Hell80 Hellman, M. E. "A cryptanalytic time-memory tradeoff," *IEEE Trans. Inf. Theory* **IT-26** (July 1980), 401-406.

INWG78 "A proposal of an Internetwork end to end protocol," in *Proc. Symp. Computer Network Protocols*, University of Liege, Belgium, Feb. 1978, pp. H:5-25.

Irla78 Irland, M. I. "Buffer management in a packet switch," *IEEE Trans. Commun.* **COM-26** (March 1978), 328-337.

Kamo81 Kamoun, F "A drop and throttle flow control policy for computer networks," *IEEE Trans Commun.* **COM-29** (April 1981), 444-452.

Klei78 Kleinrock, L., and Yemini, Y. "An optimal adaptive scheme for multiple access broadcast communication," in *Proc. ICC* (IEEE), 1978, pp 7.2.1-7.2.5.

Kuo81 Kuo, F. F. (Ed.) *Protocols and techniques for data communication networks*, Prentice-Hall, Englewood Cliffs, N.J , 1981.

Liu78 Liu, M. T. "Distributed loop computer networks," in *Advances in Computers*, M. C. Yovits (Ed.), Academic Press, New York, 1978, pp 163-221.

Mann78 Manning, E G. "On datagram service in public packet-switched networks," *Comput. Networks* **2** (May 1978), 79-83

Mart78 Martin, J. *Communications satellite systems*, Prentice-Hall, Englewood Cliffs, N.J , 1978

McQu74 McQuillan, J. M. "Adaptive routing algorithms for distributed computer networks," Ph.D dissertation, Div. Engineering and Applied Sciences, Harvard Univ., 1974

McQu77 McQuillan, J M., and Walden, D C. "The ARPA network design decisions," *Comput. Networks* **1** (Aug. 1977), 243-289.

Merk78a Merkle, R. C. "Secure communications over insecure channels," *Commun. ACM* **21** (April 1978), 294-299.

Merk78b Merkle, R. C., and Hellman, M. E. "Hiding information and receipts in trapdoor knapsacks," *IEEE Trans. Inf. Theory* **IT-24** (Sept. 1978), 525-530.

Merl80a Merlin, P. M., and Schweitzer, P. J. "Deadlock avoidance in store-and-forward networks—I. Store-and-forward deadlock," *IEEE Trans. Commun.* **COM-28** (March 1980), 345-354.

Merl80b Merlin, P M., and Schweitzer, P. J. "Deadlock avoidance in store-and-forward networks—II: Other deadlock types," *IEEE Trans. Commun.* **COM-28** (March 1980), 355-360.

Metc76 Metcalfe, R. M , and Boggs, D R. "Ethernet: Distributed packet switching for local computer networks," *Commun. ACM* **19** (July 1976), 395-404.

Need78 Needham, R. M , and Schroeder, M. D. "Using encryption for authentication in large networks of computers," *Commun. ACM* **21** (Dec. 1978), 993-999.

Need79 Needham, R M "System aspects of the Cambridge ring," in *Proc. 7th Symp. Operating Systems, Principles* (ACM), 1979, pp. 82-85.

Pete61 Peterson, W. W., and Brown, D. T. "Cyclic codes for error detection," *Proc IRE* **49** (Jan. 1961), 228-235.

Pope79 Popek, G. J., and Kline, C. S. "Encryption and secure computer networks," *Comput. Surveys* **11** (Dec 1979), 331-356.

Post80 Postel, J. B. "Internetwork protocol approaches," *IEEE Trans. Commun* **COM-28** (April 1980), 604-611.

Rive78 Rivest, R. L., Shamir, A , and Adleman, L. "A method for obtaining digital signatures and public key cryptosystems," *Commun. ACM* **21** (Feb. 1978), 120-126

Robe73 Roberts, L. G "Dynamic allocation of satellite capacity through packet reservation," in *1973 Nat. Computer Conf.*, AFIPS Press, Arlington, Va., pp. 711-716.

Roth77 Rothauser, E. H., and Wild, D. "MLMA—a collision-free multi-access method," *Proc IFIP Congr 77*, (IFIP) (1977), 431-436.

Rudi76 Rudin, H. "On routing and delta routing: A taxonomy and performance comparison of techniques for packet-switched networks" *IEEE Trans. Commun.* **COM-24** (Jan. 1976), 43-59.

Rybc80 Rybczynski, A. "X.25 interface and end-to-end virtual circuit service characteristics," *IEEE Trans. Commun.* **COM-28** (April 1980), 500-510.

Salt78 Saltzer, J. H. "On digital signatures," *Oper. Syst Rev* **12** (April 1978), 12-14

Schi78 Schicker, P., and Duenki, A "The virtual termınal definition," *Comput. Networks* **2** (Dec. 1978), 429-441

SCHO76 SCHOLL, M "Multiplexıng techniques for data transmission over packet switched radio systems," Ph.D. dıssertatıon, Computer Scıence Dep., UCLA, 1976.

SCHW80 SCHWARTZ, M., AND STERN, T. E. "Routing technıques used in computer communication networks," *IEEE Trans. Commun.* **COM-28** (April 1980), 539-552.

SEGA81 SEGALL, A "Advances in verifiable faılsafe routıng procedures," *IEEE Trans Commun.* **COM-29** (April 1981), 491-497

SHAM79 SHAMIR, A. "How to share a secret," *Commun ACM* **22** (Nov. 1979), 612-613

SHAM80 SHAMIR, A , AND ZIPPEL, R. "On the security of the Merkle-Hellman cryptographic scheme," *IEEE Trans. Inf. Theory* **IT-26** (May 1980), 339-340.

SHAP77 SHAPLEY, D., AND KOLATA, G. B. "Cryptology: Scientists puzzle over threat to open research, publıcation," *Scıence* **197** (Sept. 30, 1977), 1345-1349.

SHOC81 SHOCH, J. "An annotated bıbliography on local compuer networks," Xerox Tech. Rep., Xerox PARC, Aprıl 1980.

SLOA75 SLOANE, N. J A. *A short course on error correctıng codes,* Springer-Verlag, Berlın and New York, 1975.

SUGA79 SUGARMAN, R. M. "On foiling computer crıme," *IEEE Spectrum* **16** (July 1979), 31-32.

SUNS78 SUNSHINE, C. A., AND DALAL, Y. K "Connectıon management in transport protocols," *Comput. Networks* **2** (Dec. 1978), 454-473.

TANE81 TANENBAUM, A. S. *Computer networks,* Prentıce-Hall, Englewood Cliffs, N.J , 1981.

TOML75 TOMLINSON, R. S. "Selecting sequence numbers," in *Proc. ACM SIGCOMM/SIGOPS Interprocess Communıcatıon Workshop* (ACM) (1975), pp. 11-23.

WILK79 WILKES, M. V., AND WHEELER, D. J. "The Cambridge digital communication rıng," ın *Proc. Local Area Communıcatıon Network Symp.*, Mitre Corp and NBS (1979), pp. 47-61.

YEMI79 YEMINI, Y., AND COHEN, D. "Some issues in distributed process communication," in *Proc. 1st Int. Conf Dıstributed Computer Systems* (IEEE), 1979, pp. 199-203.

ZIMM80 ZIMMERMANN, H. "OSI reference model—the ISO model of architecture for open systems ınterconnection," *IEEE Trans. Commun.* **COM-28** (Aprıl 1980), 425-432.

Received January 1981, fınal revısıon accepted September 1981