Second Edition

Data Networks

DIMITRI BERTSEKAS

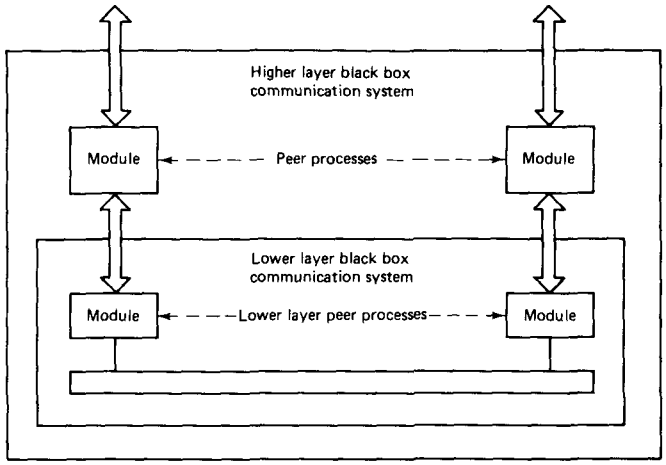
Massachusetts Institute of Technology

ROBERT GALLAGER

Massachusetts Institute of Technology

PRENTICE HALL, Englewood Cliffs, New Jersey 07632

1



Introduction and Layered Network Architecture

1.1 HISTORICAL OVERVIEW

Primitive forms of data networks have a long history, including the smoke signals used by primitive societies, and certainly including nineteenth-century telegraphy. The messages in these systems were first manually encoded into strings of essentially binary symbols, and then manually transmitted and received. Where necessary, the messages were manually relayed at intermediate points.

A major development, in the early 1950s, was the use of communication links to connect central computers to remote terminals and other peripheral devices, such as printers and remote job entry points (RJEs) (see Fig. 1.1). The number of such peripheral devices expanded rapidly in the 1960s with the development of time-shared computer systems and with the increasing power of central computers. With the proliferation of remote peripheral devices, it became uneconomical to provide a separate long-distance communication link to each peripheral. Remote multiplexers or concentrators were developed to collect all the traffic from a set of peripherals in the same area and to send it on a single link to the central processor. Finally, to free the central processor from handling all this communication, special processors called *front ends* were developed to

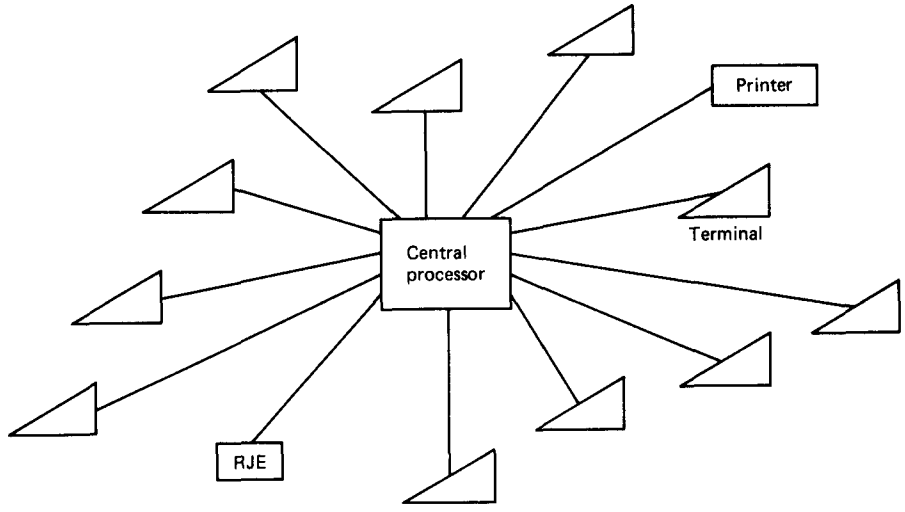


Figure 1.1 Network with one central processor and a separate communication link to each device.

control the communication to and from all the peripherals. This led to the more complex structure shown in Fig. 1.2. The communication is automated in such systems, in contrast to telegraphy, for example, but the control of the communication is centrally exercised at the computer. While it is perfectly appropriate and widely accepted to refer to such a system as a data network or computer communication network, it is simpler to view it as a computer with remote peripherals. Many of the interesting problems associated with data networks, such as the distributed control of the system, the relaying of messages over multiple communication links, and the sharing of communication links between many users and processes, do not arise in these centralized systems.

The ARPANET and TYMNET, introduced around 1970, were the first large-scale, general-purpose data networks connecting geographically distributed computer systems, users, and peripherals. Figure 1.3 shows such networks. Inside the "subnet" are a set of nodes, various pairs of which are connected by communication links. Outside the subnet are the various computers, data bases, terminals, and so on, that are connected via the subnet. Messages originate at these external devices, pass into the subnet, pass from node to node on the communication links, and finally pass out to the external recipient. The nodes of the subnet, usually computers in their own right, serve primarily to route the messages through the subnet. These nodes are sometimes called IMPs (interface message processors) and sometimes called switches. In some networks (*e.g.*, DECNET), nodes in the subnet might be physically implemented within the external computers using the network. It is helpful, however, to view the subnet nodes as being logically distinct from the external computers.

It is important to observe that in Figs. 1.1 and 1.2 the computer system is the center of the network, whereas in Fig. 1.3 the subnet (*i.e.*, the communication part of the network) is central. Keeping this picture of external devices around a communication

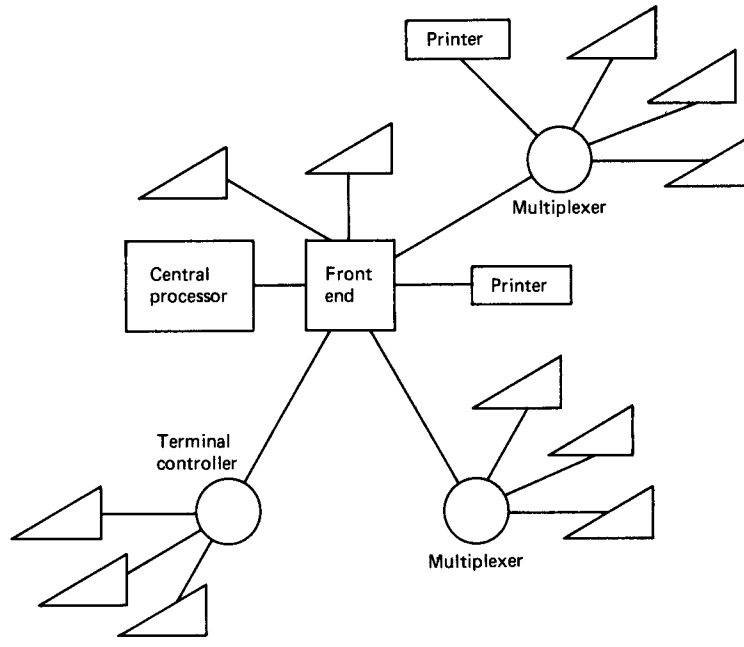


Figure 1.2 Network with one central processor but with shared communication links to devices.

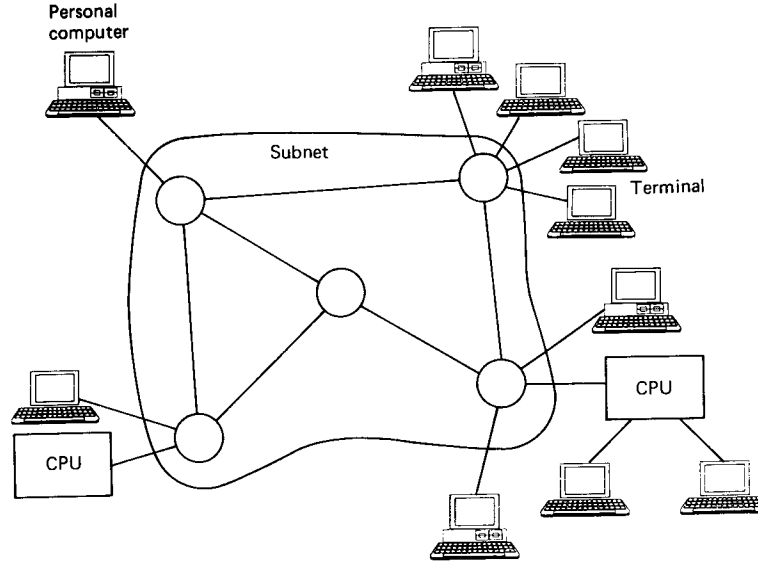


Figure 1.3 General network with a subnet of communication links and nodes. External devices are connected to the subnet via links to the subnet nodes.

subnet in mind will make it easier both to understand network layering later in this chapter and to understand the issues of distributed network control throughout the book.

The subnet shown in Fig. 1.3 contains a somewhat arbitrary placement of links between the subnet nodes. This arbitrary placement (or arbitrary topology as it is often called) is typical of wide area networks (*i.e.*, networks covering more than a metropolitan area). Local area networks (*i.e.*, networks covering on the order of a square kilometer or less) usually have a much more restricted topology, with the nodes typically distributed on a bus, a ring, or a star.

Since 1970 there has been an explosive growth in the number of wide area and local area networks. Many examples of these networks are discussed later, including as wide area networks, the seminal ARPANET and TYMNET, and as local area networks, Ethernets and token rings. For the moment, however, Fig. 1.3 provides a generic model for data networks.

With the multiplicity of different data networks in existence in the 1980s, more and more networks have been connected via gateways and bridges so as to allow users of one network to send data to users of other networks (see Fig. 1.4). At a fundamental level, one can regard such a network of networks as simply another network, as in Fig. 1.3, with each gateway, bridge, and subnet node of each constituent network being a subnet node of the overall network. From a more practical viewpoint, a network of networks is much more complex than a single network. The problem is that each constituent subnet has its own conventions and control algorithms (*i.e.*, protocols) for handling data, and the gateways and bridges must deal with this inhomogeneity. We discuss this problem later after developing some understanding of the functioning of individual subnets.

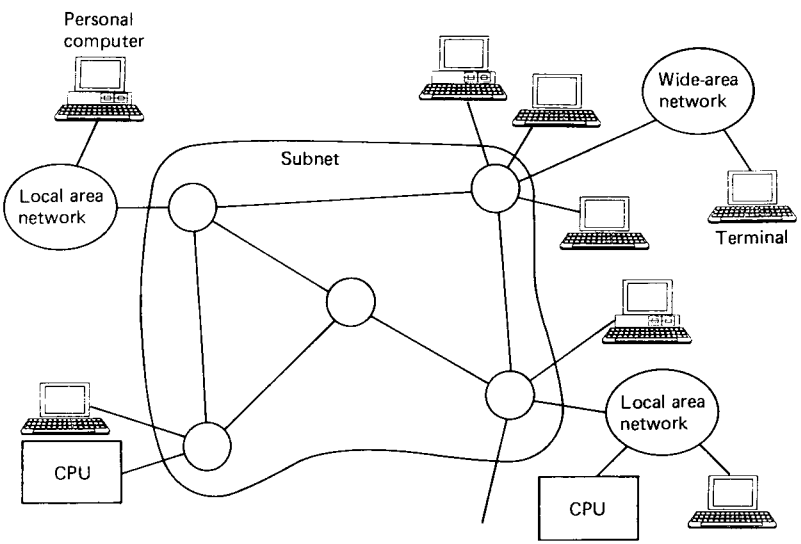


Figure 1.4 Network of interconnected networks. Individual wide area networks (WANs) and local networks (LANs) are connected via bridges and gateways.

In the future, it is likely that data networks, the voice network, and perhaps cable TV networks will be far more integrated than they are today. Data can be sent over the voice network today, as explained more fully in Section 2.2, and many of the links in data networks are leased from the voice network. Similarly, voice can be sent over data networks. What is envisioned for the future, however, is a single integrated network, called an integrated services digital network (ISDN), as ubiquitous as the present voice network. In this vision, offices and homes will each have an access point into the ISDN that will handle voice, current data applications, and new applications, all with far greater convenience and less expense than is currently possible. ISDN is currently available in some places, but it is not yet very convenient or inexpensive. Another possibility for the future is called broadband ISDN. Here the links will carry far greater data rates than ISDN and the network will carry video as well as voice and data. We discuss the pros and cons of this in Section 2.10.

1.1.1 Technological and Economic Background

Before data networks are examined, a brief overview will be given, first, of the technological and economic factors that have led to network development, and second, of the applications that require networks. The major driving force in the rapid advances in computers, communication, and data networks has been solid-state technology and in particular the development of very large scale integration (VLSI). In computers, this has led to faster, less expensive processors; faster, larger, less expensive primary memory; and faster, larger, less expensive bulk storage. The result has been the lowering of the cost of computation by roughly a factor of 2 every two years. This has led to a rapid increase in the number of cost effective applications for computers.

On the other hand, with the development of more and more powerful microprocessor chips, there has been a shift in cost effectiveness from large time-shared computer facilities to small but increasingly powerful personal computers and workstations. Thus, the primary growth in computation is in the number of computer systems rather than in the increasing power of a small number of very large computer systems.

This evolution toward many small but powerful computers has had several effects on data networks. First, since individual organizations use many computers, there is a need for them to share each other's data, thus leading to the network structures of Figs. 1.3 and 1.4. (If, instead, the evolution had been toward a small number of ever more powerful computer systems, the structure of Fig. 1.2 would still predominate.) Second, since subnet nodes are small computers, the cost of a given amount of processing within a subnet has been rapidly decreasing. Third, as workstations become more powerful, they deal with larger blocks of data (*e.g.*, high-resolution graphics) and the data rates of present-day wide area data networks are insufficient to transmit such blocks with acceptable delay.

The discussion of computational costs above neglects the cost of software. While the art of software design has been improving, the improvement is partly counterbalanced by the increasing cost of good software engineers. When software can be replicated, however, the cost per unit goes down inversely with the number of replicas. Thus,

even though software is a major cost of a new computer system, the increasing market decreases its unit cost. Each advance in solid-state technology decreases cost and increases the performance of computer systems; this leads to an increase in market, thus generating decreased unit software costs, leading, in a feedback loop, to further increases in market. Each new application, however, requires new specialized software which is initially expensive (until a market develops) and which requires a user learning curve. Thus, it is difficult to forecast the details of the growth of both the computer market and the data network market.

1.1.2 Communication Technology

The communication links for wide area networks are usually leased from the facilities of the voice telephone network. In Section 2.2 we explain how these physical links are used to transmit a fixed-rate stream of binary digits. The rate at which a link transmits binary digits is usually referred to as the data rate, capacity, or speed of the link, and these data rates come in standard sizes. Early networks typically used link data rates of 2.4, 4.8, 9.6, and 56 kilobits/sec, whereas newer networks often use 64 kilobits/sec, 1.5 megabits/sec, and even 45 megabits/sec. There are major economies of scale associated with higher link speeds; for example, the cost of a 1.5 megabit/sec link is about six times that of a 64 kilobit/sec link, but the data rate is 24 times higher. This makes it economically advantageous to concentrate network traffic on a relatively small set of high-speed links. (This effect is seen in Fig. 1.2 with the use of multiplexers or concentrators to share communication costs.)

One result of sharing high-speed (*i.e.*, high data rate) communication links is that the cost of sending data from one point to another increases less than linearly with the geographic separation of the points. This occurs because a user with a long communication path can share one or more high-speed links with other users (thus achieving low cost per unit data) over the bulk of the path, and use low-speed links (which have high cost per unit data) only for local access to the high-speed links.

Estimating the cost of transmission facilities is highly specialized and complex. The cost of a communication link depends on whether one owns the facility or leases it; with leasing, the cost depends on the current competitive and regulatory situation. The details of communication cost will be ignored in what follows, but there are several overall effects of these costs that are important.

First, for wide area data networks, cost has until recently been dominated by transmission costs. Thus, it has been desirable to use the communication links efficiently, perhaps at added computational costs. As will be shown in Section 1.3, the sporadic nature of most data communication, along with the high cost of idle communication links, led to the development of packet data networks.

Second, because of the gradual maturing of optical fiber technology, transmission costs, particularly for high data rate links, are dropping at an accelerating rate which is expected to continue well into the future. The capacity of a single optical fiber using today's technology is 10^9 to 10^{10} bits/sec, and in the future this could rise to 10^{14} or more. In contrast, all the voice and data traffic in the United States amounts to about

10^{12} bits/sec. Optical fiber is becoming widespread in use and is expected to be the dominant mode of transmission in the future. One consequence of this is that network costs are not expected to be dominated by transmission costs in the future. Another consequence is that network link capacities will increase dramatically; as discussed later, this will change the nature of network applications.

Third, for local area networks, the cost of a network has never been dominated by transmission costs. Coaxial cable and even a twisted pair of wires can achieve relatively high-speed communication at modest cost in a small geographic area. The use of such media and the desire to avoid relatively expensive switching have led to a local area network technology in which many nodes share a common high-speed communication medium on a shared multiaccess basis. This type of network structure is discussed in Chapter 4.

1.1.3 Applications of Data Networks

With the proliferation of computers referred to above, it is not difficult to imagine a growing need for data communication. A brief description of several applications requiring communication will help in understanding the basic problems that arise with data networks.

First, there are many applications centered on remote accessing of central storage facilities and of data bases. One common example is that of a local area network in which a number of workstations without disk storage use one or more common file servers to access files. Other examples are the information services and financial services available to personal computer users. More sophisticated examples, requiring many interactions between the remote site and the data base and its associated programs, include remote computerized medical diagnoses and remote computer-aided education. In some of these examples, there is a cost trade-off between maintaining the data base wherever it might be required and the communication cost of remotely accessing it as required. In other examples, in which the data base is rapidly changing, there is no alternative to communication between the remote sites and the central data base.

Next, there are many applications involving the remote updating of data bases, perhaps in addition to accessing the data. Airline reservation systems, automatic teller machines, inventory control systems, automated order entry systems, and word processing with a set of geographically distributed authors provide a number of examples. Weather tracking systems and military early warning systems are larger-scale examples. In general, for applications of this type, there are many geographically separated points at which data enter the system and often many geographically separated points at which outputs are required. Whether the inputs are processed and stored at one point (as in Figs. 1.1 and 1.2) or processed and stored at many points (as in Fig. 1.3), there is a need for a network to collect the inputs and disseminate the outputs. In any data base with multiple users there is a problem maintaining consistency (*e.g.*, two users of an airline reservation system might sell the same seat on some flight). In geographically distributed systems these problems are particularly acute because of the networking delays.

The communication requirements for accessing files and data bases have been increasing rapidly in recent years. Part of the reason for this is just the natural growth

of an expanding field. Another reason is that workstations are increasingly graphics oriented, and transmitting a high-resolution image requires millions of bits. Another somewhat related reason is that the link capacities available in local area networks have been much larger than those in wide area networks. As workstation users get used to sending images and large files over local area nets, they expect to do the same over wide area networks. Thus the need for increased link capacity for wide area networks is particularly pressing.

Another popular application is electronic mail between the human users of a network. Such mail can be printed, read, filed, forwarded to other individuals, perhaps with added comments, or read by the addressee at different locations. It is clear that such a service has many advantages over postal mail in terms of delivery speed and flexibility.

In comparison with facsimile, which has become very popular in recent years, electronic mail is more economical, has the flexibility advantages above, and is in principle more convenient for data already stored in a computer. Facsimile is far more convenient for data in hard-copy form (since the hard copy is fed directly into the facsimile machine). It appears clear, however, that the recent popularity of facsimile is due to the fact that it is relatively hassle-free, especially for the occasional or uninitiated user. Unfortunately, electronic mail, and more generally computer communication, despite all the cant about user friendliness, is full of hassles and pitfalls for the occasional or uninitiated user.

There is a similar comparison of electronic mail with voice telephone service. Voice service, in conjunction with an answering machine or voice mail service, in principle has most of the flexibility of electronic mail except for the ability to print a permanent record of a message. Voice, of course, has the additional advantage of immediate two-way interaction and of nonlinguistic communication via inflection and tone. Voice communication is more expensive, but requires only a telephone rather than a telephone plus computer.

As a final application, one might want to use a remote computer system for some computational task. This could happen as a means of load sharing if the local computer is overutilized. It could also arise if there is no local computer, if the local computer is inoperational, or the remote computer is better suited to the given task. Important special cases of the latter are very large problems that require supercomputers. These problems frequently require massive amounts of communication, particularly when the output is in high resolution graphic form. Present-day networks, with their limited link speeds, are often inadequate for these tasks. There are also "real-time" computational tasks in which the computer system must respond to inputs within some maximum delay. If such a task is too large for the local computer, it might be handled by a remote supercomputer or by a number of remote computers working together. Present-day networks are also often inadequate for the communication needs of these tasks.

It will be noted that all the applications above could be satisfied by a network with centralized computer facilities as in Fig. 1.1 or 1.2. To see this, simply visualize moving all the large computers, data bases, and subnet nodes in the network of Fig. 1.3 to one centralized location, maintaining links between all the nodes previously connected. The central facilities would then be connected by short communication lines rather than long, but aside from some changes in propagation delays, the overall network would be

unchanged. Such a geographically centralized but logically distributed structure would both allow for shared memory between the computers and for centralized repair. Why, then, are data networks with geographically distributed computational and data base facilities growing so quickly in importance? One major reason is the cost and delay of communication. With distributed computers, many computational tasks can be handled locally. Even for remote tasks, communication costs can often be reduced significantly by some local processing. Another reason is that organizations often acquire computers for local automation tasks, and only after this local automation takes place does the need for remote interactions arise. Finally, organizations often wish to have control of their own computer systems rather than be overly dependent on the pricing policies, software changes, and potential security violations of a computer utility shared with many organizations.

Another advantage often claimed for a network with distributed computational facilities is increased reliability. For the centralized system in Fig. 1.2 there is some truth to this claim, since the failure of a communication link could isolate a set of sites from all access to computation. For the geographically centralized but logically distributed network, especially if there are several disjoint paths between each pair of sites, the failure of a communication link is less critical and the question of reliability becomes more complex. If all the large computers and data bases in a network were centralized, the network could be destroyed by a catastrophe at the central site. Aside from this possibility, however, a central site can be more carefully protected and repairs can be made more quickly and easily than with distributed computational sites. Other than these effects, there appears to be no reason why geographically distributed computational facilities are inherently more or less reliable than geographically centralized (but logically distributed) facilities. At any rate, the main focus in what follows will be on networks as in Figs. 1.3 and 1.4, where the communication subnet is properly viewed as the center of the entire network.

1.2 MESSAGES AND SWITCHING

1.2.1 Messages and Packets

A message in a data network corresponds roughly to the everyday English usage of the word. For example, in an airline reservation system, we would regard a request for a reservation, including date, flight number, passenger names, and so on, as a message. In an electronic mail system, a message would be a single document from one user to another. If that same document is then forwarded to several other users, we would sometimes want to regard this forwarding as several new messages and sometimes as forwarding of the same message, depending on the context. In a file transfer system, a message would usually be regarded as a file. In an image transmission system (*i.e.*, pictures, figures, diagrams, etc.), we would regard a message as an image. In an application requiring interactive communication between two or more users, a message would be one unit of communication from one user to another. Thus, in an interactive transaction,

user 1 might send a message to user 2, user 2 might reply with a message to 1, who might then send another message to 2, and so forth until the completion of the overall transaction. The important characteristic of a message is that from the standpoint of the network users, it is a single unit of communication. If a recipient receives only part of a message, it is usually worthless.

It is sometimes necessary to make a distinction between a message and the representation of the message. Both in a subnet and in a computer, a message is usually represented as a string of binary symbols, 0 or 1. For brevity, a binary symbol will be referred to as a *bit*. When a message goes from sender to recipient, there can be several transformations on the string of bits used to represent the message. Such transformations are sometimes desirable for the sake of data compression and sometimes for the sake of facilitating the communication of the message through the network. A brief description of these two purposes follows.

The purpose of data compression is to reduce the length of the bit string representing the message. From the standpoint of information theory, a message is regarded as one of a collection of possible messages, with a probability distribution on the likelihood of different messages. Such probabilities can only be crudely estimated, either a priori or adaptively. The idea, then, is to assign shorter bit strings to more probable messages and longer bit strings to less probable messages, thus reducing the expected length of the representation. For example, with text, one can represent common letters in the alphabet (or common words in the dictionary) with a small number of bits and represent unusual letters or words with more bits. As another example, in an airline reservation system, the common messages have a very tightly constrained format (date, flight number, names, etc.) and thus can be very compactly represented, with longer strings for unusual types of situations. Data compression will be discussed more in Chapter 2 in the context of compressing control overhead. Data compression will not be treated in general here, since this topic is separable from that of data networks, and is properly studied in its own right, with applications both to storage and point-to-point communication.

Transforming message representations to facilitate communication, on the other hand, is a central topic for data networks. In subsequent chapters, there are many examples in which various kinds of control overhead must be added to messages to ensure reliable communication, to route the message to the correct destination, to control congestion, and so on. It will also be shown that transmitting very long messages as units in a subnet is harmful in several ways, including delay, buffer management, and congestion control. Thus, messages represented by long strings of bits are usually broken into shorter bit strings called *packets*. These packets can then be transmitted through the subnet as individual entities and reassembled into messages at the destination.

The purpose of a subnet, then, is to receive packets at the nodes from sites outside the subnet, then transmit these packets over some path of communication links and other nodes, and finally deliver them to the destination sites. The subnet must somehow obtain information about where the packet is going, but the meaning of the corresponding message is of no concern within the subnet. To the subnet, a packet is simply a string of bits that must be sent through the subnet reliably and quickly. We return to this issue in Section 1.3.

1.2.2 Sessions

Messages between two users usually occur as a sequence in some larger transaction; such a message sequence (or, equivalently, the larger transaction) is called a *session.* For example, updating a data base usually requires an interchange of several messages. Writing a program at a terminal for a remote computer usually requires many messages over a considerable time period. Typically, a setup procedure (similar to setting up a call in a voice network) is required to initiate a session between two users, and in this case a session is frequently called a *connection.* In other networks, no such setup is required and each message is treated independently; this is called a *connectionless* service. The reasons for these alternatives are discussed later.

From the standpoint of network users, the messages within a session are typically triggered by particular events. From the standpoint of the subnet, however, these message initiation times are somewhat arbitrary and unpredictable. It is often reasonable, for subnet purposes, to model the sequence of times at which messages or packets arrive for a given session as a random process. For simplicity, these arrivals will usually be modeled as occurring at random points in time, independently of each other and of the arrivals for other sessions. This type of arrival process is called a *Poisson* process and is defined and discussed in Section 3.3. This model is not entirely realistic for many types of sessions and ignores the interaction between the messages flowing in the two directions for a session. However, such simple models provide insight into the major trade-offs involved in network design, and these trade-offs are often obscured in more realistic and complex models.

Sometimes it will be more convenient to model message arrivals within a session by an on/off flow model. In such a model, a message is characterized by a sequence of bits flowing into the subnet at a given rate. Successive message arrivals are separated by random durations in which no flow enters the network. Such a model is appropriate, for example, for voice sessions and for real-time monitoring types of applications. When voice is digitized (see Section 2.2), there is no need to transmit when the voice is silent, so these silence periods correspond to the gaps in an on/off flow model. One might think that there is little fundamental difference between a model using point arrivals for messages and a model using on/off flow. The output from point message arrivals, followed by an access line of fixed rate, looks very much like an on/off flow (except for the possibilitity that one message might arrive while another is still being sent on the access line). The major difference between these models, however, is in the question of delay. For sessions naturally modeled by point message arrivals (*e.g.*, data base queries), one is usually interested in delay from message arrival to the delivery of the entire message (since the recipient will process the entire message as a unit). For sessions naturally modeled by flow (such as digitized voice), the concept of a message is somewhat artificial and one is usually interested in the delay experienced by the individual bits within the flow. It appears that the on/off flow model is growing in importance and is particularly appropriate for ISDN and broadband ISDN networks. Part of the reason for this growth is the prevalence of voice in ISDN and voice and video in broadband ISDN. An-

other reason, which will be more clear later, is that very long messages, which will be prevalent with ISDN, are probably better treated in the subnet as flows than as point arrivals.

To put this question of modeling message arrivals for a session in a more pragmatic way, note that networks, particularly wide area networks built around a subnet as in Fig. 1.3, generally handle multiple applications. Since the design and implementation of a subnet is a time-consuming process, and since applications are rapidly changing and expanding, subnets must be designed to handle a wide variety of applications, some of which are unknown and most of which are subject to change. Any complex model of message arrivals for sessions is likely to be invalid by the time the network is used. This point of view, that subnets must be designed to work independently of the fine details of applications, is discussed further in Section 1.3.

At this point we have a conceptual view, or model, of the function of a subnet. It will provide communication for a slowly varying set of sessions; within each session, messages of some random length distribution arrive at random times according to some random process. Since we will largely ignore the interaction between the two directions of message flow for a session, we shall usually model a two-way session as two one-way sessions, one corresponding to the message flow in one direction and the other in the opposite direction. In what follows we use the word *session* for such one-way sessions. In matters such as session initiation and end-to-end acknowledgment, distinctions are made between two-way and one-way sessions.

In principle a session could involve messages between more than two users. For example, one user could broadcast a sequence of messages to each of some set of other users, or the messages of each user in the set could be broadcast to each of the other users. Such sessions might become important in the future, especially for broadband ISDN, with applications such as video conferencing and television broadcast. We will not discuss such applications in any detail, but instead will simply model multiuser sessions as a multiplicity of one-way two-user sessions.

Although the detailed characteristics of different kinds of applications will not be examined, there are some gross characteristics of sessions that must be kept in mind. The most important are listed:

1. *Message arrival rate and variability of arrivals.* Typical arrival rates for sessions vary from zero to more than enough to saturate the network. Simple models for the variability of arrivals include Poisson arrivals, deterministic arrivals (*i.e.*, a fixed time interval from each message to the next message), and uniformly distributed arrivals (*i.e.*, the time interval between successive messages has a uniform probability density between some minimum and maximum interval).
2. *Session holding time.* Sometimes (as with electronic mail) a session is initiated for a single message. Other sessions last for a working day or even permanently.
3. *Expected message length and length distribution.* Typical message lengths vary roughly from a few bits to 10^9 bits, with file transfer applications at the high end and interactive sessions from a terminal to a computer at the low end. Simple models for length distribution include an exponentially decaying probability density, a uniform

probability density between some minimum and maximum, and fixed length. As mentioned above, long messages are becoming much more common because of graphics and long file transfers.

4. *Allowable delay.* The allowable expected delay varies from about 10 msec for some real-time control applications to 1 sec or less for interactive terminal to computer applications, to several minutes or more for some file transfer applications. In other applications, there is a maximum allowable delay (in contrast to expected delay). For example, with packetized voice, fixed-length segments of the incoming voice waveform are encoded into packets at the source. At the destination, these packets must be reconverted into waveform segments with some fixed overall delay; any packet not received by this time is simply discarded. As described above, delay is sometimes of interest on a message basis and sometimes, in the flow model, on a bit basis.
5. *Reliability.* For some applications, all messages must be delivered error-free. For example, in banking applications, in transmission of computer programs, or in file transfers, a single bit error in a message can have serious consequences. In other applications, such as electronic mail, all messages must be delivered, but an occasional bit error in a message can usually be visually corrected by the reader. Finally, in other applications, both occasional bit errors and occasional loss of entire packets or messages are allowable. For example, in distributed sensor systems, messages are sometimes noisy when transmitted, and occasional lost messages are soon replaced with more up-to-date messages. For packetized voice, the occasional loss (or late delivery) of a packet or an occasional bit error simply increases the noisiness of the received voice signal. It should be noted, however, that the use of data compression for packetized voice and other applications greatly increases the need for error-free communication.
6. *Message and packet ordering.* The packets within a message must either be maintained in the correct order going through the network or restored to the correct order at some point. For many applications (such as updating data bases), messages must also be delivered in the correct order, whereas for other applications, message order is unimportant. The question of where to handle reliability and message ordering (*i.e.*, at the external sites or within the subnet or both) is an important design issue. This is discussed in Section 2.8.

In keeping all these characteristics in mind, it is often helpful to focus on four types of applications which lie somewhat at the extreme points and which do not interact very well together in subnets. One is interactive terminal to computer sessions, in which messages are short, the message rate is low, the delay requirement is moderately stringent, and the need for reliability is high. Another is file transfer sessions, in which the messages are very long, the message arrival rate is typically low, the delay requirement is very relaxed, and the need for reliability is very high. The third is high-resolution graphics, in which the messages are again long, sometimes up to 10^9 bits, the delay requirement is stringent, and the arrival rate is low. The fourth is packetized voice. Here the concept of a message is not very useful, but the packets are short, the packet arrival rate is high,

the maximum delay requirement is stringent, and the need for reliability is rather low. A network that can handle all these applications together will probably not have too much difficulty with the other applications of interest.

1.2.3 Circuit Switching and Store-and-Forward Switching

There are two general approaches, known as circuit switching and store-and-forward switching, that can be used within a subnet to transmit the traffic for the various sessions. A brief overview will be given of the circuit switching approach, followed by the reason why this approach leads to inefficient utilization of the communication channels for many types of sessions. Next, an overview of the store-and-forward approach will be given, showing how it overcomes the above inefficiency.

For the circuit switching approach, when a session s is initiated, it is allocated a given transmission rate r_s in bits per second (this could be different in the two directions of a two-way session, but we focus on a one-way session here). A path is then created from the transmitting site through the subnet and to the destination site. Each communication link on this path then allocates a portion r_s of its total transmission capacity in the given direction for that session. This allocation of transmission rates to different sessions on a communication link is usually done by time-division multiplexing (TDM) or frequency-division multiplexing (FDM), but the details of that are explained in Section 2.1. What is important is that the sum of the rates for all the sessions using a link cannot exceed the total capacity of the link. Thus, if a communication link is fully allocated to existing sessions, a new session cannot use that link. If no path can be found using links with at least r_s bits/sec of unused rate, the new session must be rejected (*i.e.*, given a busy signal). The other important point is that once the session has been successfully initiated, it has a guaranteed transmission rate r_s through the network. The nodes then simply take the incoming bit stream for a given session off the incoming link and switch it to the allocated portion of the outgoing link. This type of switching is quite similar to the well-developed technology for switching in the telephone network. In the telephone network, however, each session is allocated the same transmission rate, whereas in a data network, the required transmission rates are different and vary over a wide range.

Circuit switching is rarely used for data networks. In the past, the reason for this has had nothing to do with the potential complexity of the switching, but rather, as we now explain, has been because of very inefficient use of the links. Typical data sessions tend to have short bursts of high activity followed by lengthy inactive periods; circuit switching wastes the allocated rate during these inactive periods. For a more quantitative view, let λ be the message arrival rate for a given session s. More precisely, $1/\lambda$ is the expected interarrival time between messages of s. Let $\overline{X}$ be the expected transmission time of a message over a given link in the path; that is, if $\overline{L}$ is the expected length (in bits) of messages from s, and r_s is the bit rate allocated to s, then $\overline{X} = \overline{L}/r_s$. Figure 1.5 illustrates these arrivals and transmission times.

Note from the figure that the fraction of time in which session s's portion of the link is actually transmitting messages is rather small; that portion of the link is otherwise

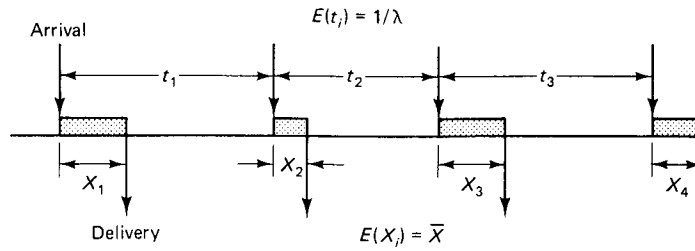


Figure 1.5 Link utilization. The expected transmission time of a message is $\overline{X}$. The expected interarrival period is $1/\lambda$. Thus, the link is used at most $\lambda\overline{X}$ of the time.

idle. It is intuitively plausible, since $1/\lambda$ is the expected interarrival time and $\overline{X}$ is the expected busy time between arrivals, that the ratio of $\overline{X}$ to $1/\lambda$ (*i.e.*, $\lambda\overline{X}$) is the fraction of time in which the portion of the link allocated to s is busy. This argument is made precise in Chapter 3. Our conclusion then is that if $\lambda\overline{X} \ll 1$, session s's portion of the link is idle most of the time (*i.e.*, inefficiently utilized).

To complete our argument about the inefficiency of circuit switching for data networks, we must relate $\overline{X}$ to the allowable expected delay T from message arrival at the source to delivery at the destination. Since $\overline{X}$ is the expected time until the last bit of the message has been sent on the first link, we must have $\overline{X} + P \leq T$, where P is the propagation delay through the network. Thus $\lambda\overline{X} < \lambda T$. If $\lambda T \ll 1$ (*i.e.*, the allowable delay is small relative to the message interarrival rate), the utilization $\lambda\overline{X}$ for the session is correspondingly small. In summary, the bit rate r_s allocated to a session must be large enough to allow message transmission within the required delay, and when $\lambda T \ll 1$, this implies inefficient utilization of the link. Sessions for which $\lambda T \ll 1$ are usually referred to as *bursty* sessions.

For many of the interactive terminal sessions carried by data networks, λT is on the order of 0.01 or less. Thus, with circuit switching, that fraction of a link allocated to such sessions is utilized at most 1% of the time. The conclusion we draw from this is that if link costs are a dominant part of the cost of a network and if bursty sessions require a dominant fraction of link capacity using circuit switching, then circuit switching is an unattractive choice for data networks. Up to the present, both the assumptions above have been valid, and for this reason, data networks have not used circuit switching. The argument above has ignored propagation delays, switching delays in the nodes, and queueing delays. (Queueing delay arises when a message from session s arrives while another message from s is in transmission.) Since these delays must be added to the link transmission time $\overline{X}$ in meeting the delay requirement T, $\overline{X}$ must often be substantially smaller than T, making circuit switching even more inefficient. While propagation and switching delays are often negligible, queueing delay is not, as shown in Chapter 3, particularly when λT is close to or exceeds 1.

In the future, it appears that link costs will become less important in the overall cost of a network. Also, with optical fiber, the marginal cost of link capacity is quite small, so that the wasted capacity of circuit switching will become less important. Finally, it appears that bursty interactive terminal traffic will grow considerably more slowly than

link capacities in the future (the reason for this is discussed later). Thus circuit switching is a feasible possibility (although not necessarily the best possibility) for networks of the future. Part of the issue here is that as link speeds increase, node processing speed must also increase, putting a premium on simple processing within the subnet. It is not yet clear whether circuit switching or store-and-forward allows simpler subnet processing at high link speeds, but store-and-forward techniques are currently receiving more attention.

In the store-and-forward approach to subnet design, each session is initiated without necessarily making any reserved allocation of transmission rate for the session. Similarly, there is no conventional multiplexing of the communication links. Rather, one packet or message at a time is transmitted on a communication link, using the full transmission rate of the link. The link is shared between the different sessions using that link, but the sharing is done on an as needed basis (*i.e.*, demand basis) rather than a fixed allocation basis. Thus, when a packet or message arrives at a switching node on its path to the destination site, it waits in a queue for its turn to be transmitted on the next link in its path.

Store-and-forward switching has the advantage over circuit switching that each communication link is fully utilized whenever it has any traffic to send. In Chapter 3, when queueing is studied, it will be shown that using communication links on a demand basis often markedly decreases the delay in the network relative to the circuit switching approach. Store-and-forward switching, however, has the disadvantage that the queueing delays in the nodes are hard to control. The packets queued at a node come from inputs at many different sites, and thus there is a need for control mechanisms to slow down those inputs when the queueing delay is excessive, or even worse, when the buffering capacity at the node is about to be exceeded. There is a feedback delay associated with any such control mechanism. First, the overloaded node must somehow send the offending inputs some control information (through the links of the network) telling them to slow down. Second, a considerable number of packets might already be in the subnet heading for the given node. This is the general topic of flow control and is discussed in Chapter 6. The reader should be aware, however, that this problem is caused by the store-and-forward approach and is largely nonexistent in the circuit switching approach.

There is a considerable taxonomy associated with store-and-forward switching. *Message switching* is store-and-forward switching in which messages are sent as unit entities rather than being segmented into packets. If message switching were to be used, there would have to be a maximum message size, which essentially would mean that the user would have to packetize messages rather than having packetization done elsewhere. *Packet switching* is store-and-forward switching in which messages are broken into packets, and from the discussion above, we see that store-and-forward switching and packet switching are essentially synonymous. *Virtual circuit routing* is store-and-forward switching in which a particular path is set up when a session is initiated and maintained during the life of the session. This is like circuit switching in the sense of using a fixed path, but it is virtual in the sense that the capacity of each link is shared by the sessions using that link on a demand basis rather than by fixed allocations. *Dynamic routing* (or *datagram routing*) is store-and-forward switching in which each packet finds its own path

through the network according to the current information available at the nodes visited. Virtual circuit routing is generally used in practice, although there are many interesting intermediate positions between virtual circuit routing and dynamic routing. The general issue of routing is treated in Chapter 5.

1.3 LAYERING

Layering, or layered architecture, is a form of hierarchical modularity that is central to data network design. The concept of modularity (although perhaps not the name) is as old as engineering. In what follows, the word *module* is used to refer either to a device or to a process within some computer system. What is important is that the module performs a given function in support of the overall function of the system. Such a function is often called the *service* provided by the module. The designers of a module will be intensely aware of the internal details and operation of that module. Someone who uses that module as a component in a larger system, however, will treat the module as a "black box." That is, the user will be uninterested in the internal workings of the module and will be concerned only with the inputs, the outputs, and, most important, the functional relation of outputs to inputs (*i.e.*, the service provided). Thus, a black box is a module viewed in terms of its input–output description. It can be used with other black boxes to construct a more complex module, which again will be viewed at higher levels as a bigger black box.

This approach to design leads naturally to a hierarchy of modules in which a module appears as a black box at one layer of the hierarchy, but appears as a system of lower-layer black boxes at the next lower layer of the hierarchy (see Fig. 1.6). At the overall system level (*i.e.*, at the highest layer of the hierarchy), one sees a small collection of top-layer modules, each viewed as black boxes providing some clear-cut service. At the next layer down, each top-layer module is viewed as a subsystem of lower-layer black boxes, and so forth, down to the lowest layer of the hierarchy. As shown in Fig. 1.6, each layer might contain not only black boxes made up of lower-layer modules but also simple modules that do not require division into yet simpler modules.

As an example of this hierarchical viewpoint, a computer system could be viewed as a set of processor modules, a set of memory modules, and a bus module. A processor module could, in turn, be viewed as a control unit, an arithmetic unit, an instruction fetching unit, and an input–output unit. Similarly, the arithmetic unit could be broken into adders, accumulators, and so on.

In most cases, a user of a black box does not need to know the detailed response of outputs to inputs. For example, precisely when an output changes in response to an input is not important as long as the output has changed by the time it is to be used. Thus, modules (*i.e.*, black boxes) can be specified in terms of tolerances rather than exact descriptions. This leads to standardized modules, which leads, in turn, to the possibility of using many identical, previously designed (*i.e.*, off-the-shelf) modules in the same system. In addition, such standardized modules can easily be replaced with new, functionally equivalent modules that are cheaper or more reliable.

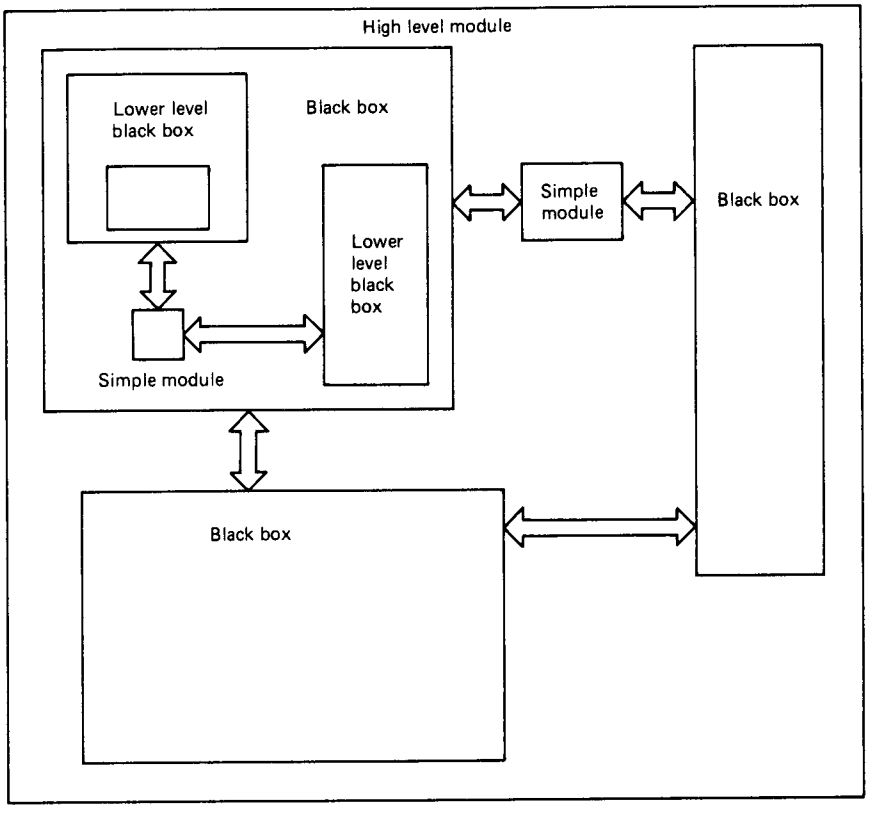


Figure 1.6 Hierarchy of nested black boxes. Each black box (except that at the lowest level) contains black boxes at a lower level, plus perhaps other modules.

All of these advantages of modularity (*i.e.*, simplicity of design; understandability; and standard, interchangeable, widely available modules) provide the motivation for a layered architecture in data networks. A layered architecture can be regarded as a hierarchy of nested modules or black boxes, as described above. Each given layer in the hierarchy regards the next lower layer as one or more black boxes which provide a specified service to the given higher layer.

What is unusual about the layered architecture for data networks is that the black boxes at the various layers are in fact distributed black boxes. The bottom layer of the hierarchy consists of the physical communication links, and at each higher layer, each black box consists of a lower-layer black box communication system plus a set of simple modules, one at each end of the lower-layer communication system. The simple modules associated with a black box at a given layer are called *peer processes* or *peer modules* (see Fig. 1.7).

In the simplest case, a black box consists of two peer processes, one at each of two nodes, and a lower-layer black box communication system connecting the two peer processes. One process communicates with its peer at the other node by placing a message

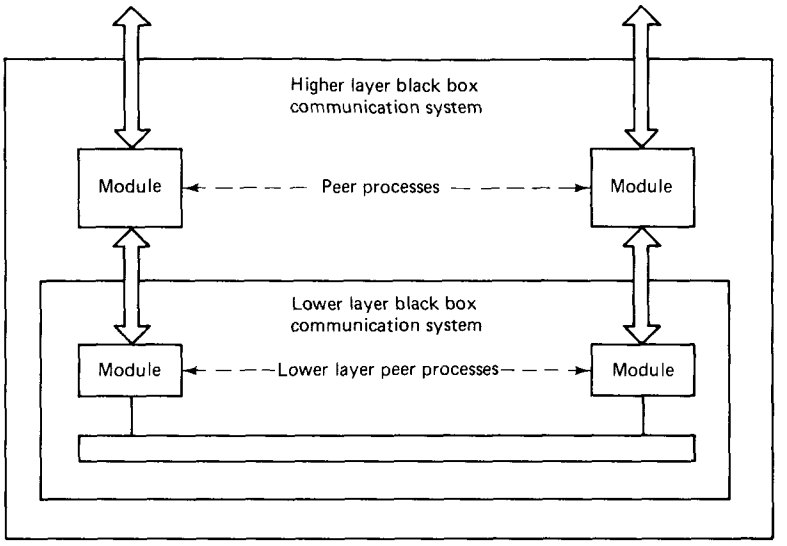


Figure 1.7 Peer processes within a black box communication system. The peer processes communicate through a lower-layer black box communication system that itself contains lower-layer peer processes.

into the lower-layer black box communication system. This lower-layer black box, as illustrated in Fig. 1.7, might in fact consist of two lower-layer peer processes, one at each of the two nodes, connected by a yet lower-layer black box communication system. As a familiar example, consider two heads of state who have no common language for communication. One head of state can then send a message to the peer head of state by a local translator, who communicates in a common language to a peer translator, who then delivers the message in the language of the peer head of state.

Note that there are two quite separate aspects to the communication between a module, say at layer n, and its layer n peer at another node. The first is the protocol (or distributed algorithm) that the peer modules use in exchanging messages or bit strings so as to provide the required functions or service to the next higher layer. The second is the specification of the precise interface between the layer n module at one node and the layer $n-1$ module at the same node through which the messages above are actually exchanged. The first aspect above is more important (and more interesting) for a conceptual understanding of the operation of a layered architecture, but the second is also vital in the actual design and standardization of the system. In terms of the previous example of communication between heads of state, the first aspect has to do with the negotiation between the heads of state, whereas the second has to do with each head of state ensuring that the translator can actually translate the messages faithfully.

Figure 1.8 illustrates such a layered architecture. The layers are those of the reference model of open systems interconnection (OSI) developed as an international standard for data networks by the International Standards Organization (ISO). Many existing networks, including SNA, DECNET, ARPANET, and TYMNET, have somewhat

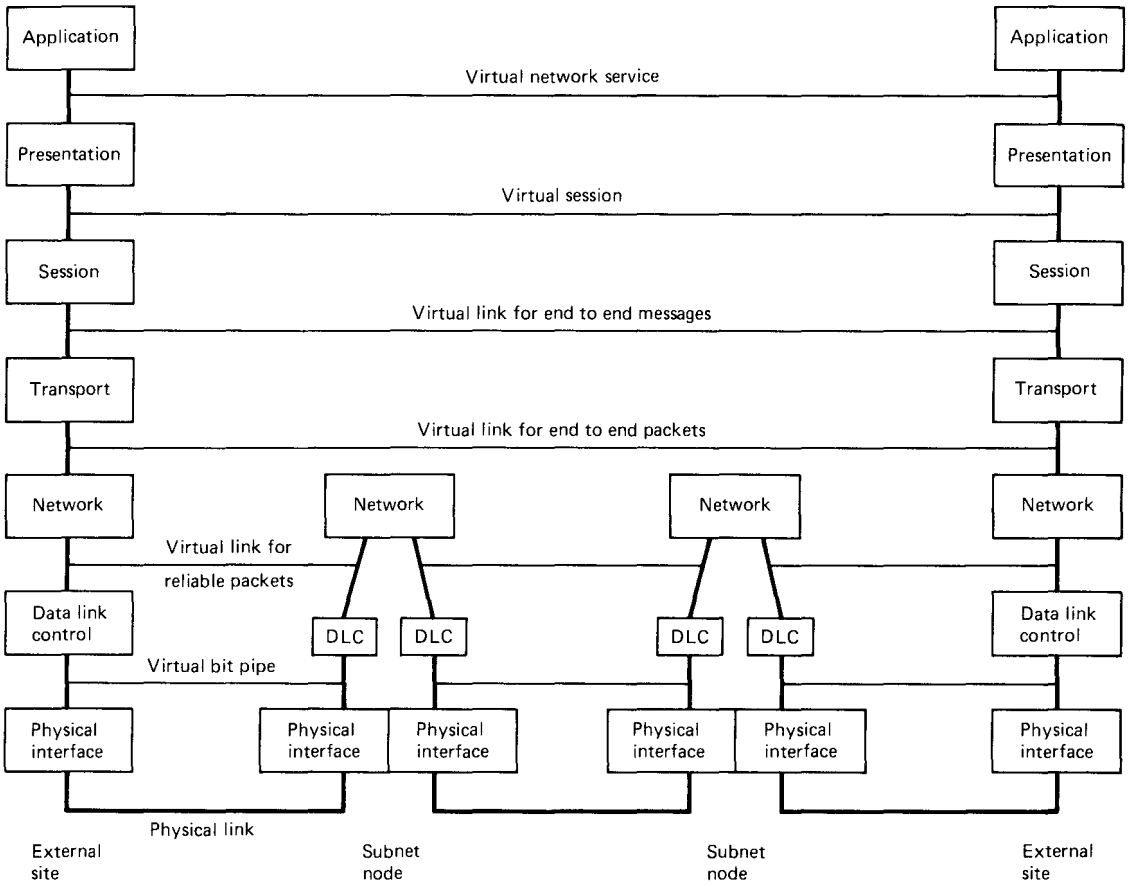


Figure 1.8 Seven-layer OSI network architecture. Each layer presents a virtual communication link with given properties to the next-higher layer.

different layers than this proposed standard. However, the OSI layers have a relatively clean structure that helps in understanding the concept of layering. Some of the variations used by these other networks are discussed later.

1.3.1 The Physical Layer

The function of the *physical layer* is to provide a virtual link for transmitting a sequence of bits between any pair of nodes (or any node and external site) joined by a physical communication channel. Such a virtual link is called a *virtual bit pipe*. To achieve this function, there is a physical interface module on each side of the communication channel whose function is to map the incoming bits from the next higher layer [*i.e.*, the data link control (DLC) layer] into signals appropriate for the channel, and at the receiving end, to map the signals back into bits. The physical interface module that

performs these mapping functions is often called a *modem* (digital data *mo*dulator and *dem*odulator). The term *modem* is used broadly here to refer to any module that performs the function above, whether or not modulation is involved; for example, if the physical communication channel is a digital link (see Section 2.2), there is nothing for the modem to do other than interface with the DLC module.

Modems and communication channels are discussed in Section 2.2. The modem designer must be aware of the detailed characteristics of the communication channel (and different modems must be designed for different types of channels). To the higher layers, however, the black box formed by the modem–channel–modem combination appears as a bit pipe with the complexities of the physical channel hidden. Even viewed as a bit pipe, however, there are a few issues that must be discussed.

The first issue has to do with the timing of the bit sequence entering the bit pipe. There are three common situations. The first is that of a *synchronous* bit pipe where bits are transmitted and received at regular intervals (*i.e.*, 1 bit per t second interval for some t). The higher-layer DLC module must supply bits at this synchronous rate whether or not it has any real data to send. The second situation is that of an *intermittent synchronous* bit pipe where the DLC module supplies bits at a synchronous rate when it has data to send and stops sending bits when there are no data to send. The third situation is that of *asynchronous characters*, usually used with personal computers and low-speed terminals. Here, keyboard characters and various control characters are mapped into fixed-length bit strings (usually, eight-bit strings according to a standard mapping from characters to bit strings known as ASCII code), and the individual character bit strings are transmitted asynchronously as they are generated.

The next issue is that of the interface between the DLC module and the modem. One would think that not many problems should exist in delivering a string of bits from one module to another, especially if they are physically close. Unfortunately, there are a number of annoying details about such an interface. For example, the module on one end of the interface might be temporarily inoperable, and when both become operable, some initialization is required to start the flow of bits. Also, for synchronous operation, one side or the other must provide timing. To make matters worse, many different manufacturers provide the modules on either end, so there is a need for standardizing the interface. In fact, there are many such standards, so many that one applauds the effort but questions the success. Two of the better known are RS-232-C and the physical layer of X.21.

The RS-232-C interface approaches the problem by providing a separate wire between the two modules for each type of control signal that might be required. These wires from the modules are joined in a standard 25-pin connector (although usually many fewer wires are required). In communication jargon, the interface is between a DCE (data communication equipment), which is the modem in this case, and a DTE (data terminal equipment), which is the DLC layer and higher layers in this case.

As an example of the interface use, suppose that the DTE wants to start sending data (either on initialization or with a new data sequence in intermittent synchronous transmission). The DTE then sends a signal to the DCE on a "request-to-send" wire. The DCE replies with a signal on the "clear-to-send" wire. The DCE also sends a signal

on the "DCE-ready" wire whenever it is operational and a signal on the "carrier detect" wire whenever it appears that the opposite modem and channel are operational. If the DTE receives all these signals (which are just level voltages), it starts to send data over the interface on the DTE-to-DCE data wire.

This interchange is a very simple example of a *protocol* or *distributed algorithm.* Each module performs operations based both on its own state and on the information received from the other module. Many less trivial protocols are developed in subsequent chapters. There are many other details in RS-232-C operation but no other new concepts.

It is sometimes helpful when focusing on the interface between the DLC module and the modem to view the wires between the modules as a physical channel and to view the DLC and modem as peer processes executing the interface protocol. To avoid confusion between the DLC module's major function as a peer process with the opposite DLC module and its lower-level function of interfacing with the modem, an extra dummy module is sometimes created (see Fig. 1.9) which exercises the interface protocol with the modem.

The X.21 physical layer interface is similar in function to RS-232-C, but it uses a smaller set of wires (eight wires are used, although there is a 15-pin connector) and is intended as an interface to a digital communication link. The idea is to avoid using a separate wire for each possible signal by doubling up on the use of wires by digital logic in the modules. The X.21 physical layer is used as the physical layer for the X.25 protocol, which is discussed in Chapter 2.

It should be clear from the above that there is a great conceptual advantage in removing the question of modem and modem interfaces from the higher-level aspects of networks. Note that this has already been done, in essence, in previous sections in referring to the number of bits per second that could be transmitted over communication links. It should also be noted, however, that modems cannot be totally segregated from network issues. For example, is it better to have a modem that transmits R bits/sec with an error rate of 10^{-6} or a modem that transmits $2R$ bits/sec with an error rate of 10^{-4}? This cannot be answered without some knowledge of how errors are eliminated at higher

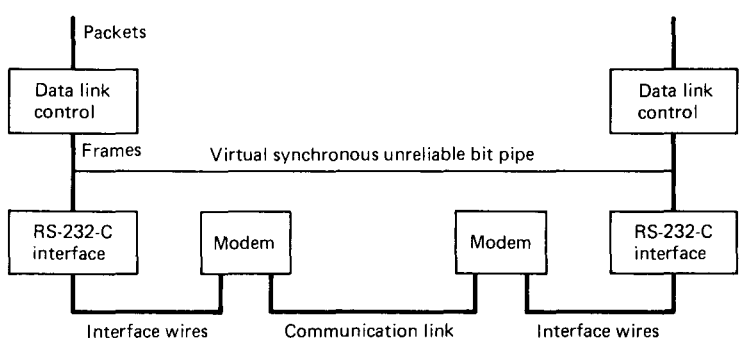


Figure 1.9 Layering with the interface between the DLC and the modem viewed as an interface over a physical medium consisting of a set of wires.

layers of the architecture. Conversely, decisions on how and where to eliminate errors at higher layers should depend on the error rate and error characteristics at the physical layer.

1.3.2 The Data Link Control Layer

The second layer in Fig. 1.8 is the *data link control (DLC) layer*. Each point-to-point communication link (*i.e.*, the two-way virtual bit pipe provided by layer 1) has data link control modules (as peer processes) at each end of the link. The customary purpose of data link control is to convert the unreliable bit pipe at layer 1 into a higher-level, virtual communication link for sending packets asynchronously but error-free in both directions over the link. From the standpoint of the DLC layer, a packet is simply a string of bits that comes from the next higher layer.

The communication at this layer is asynchronous in two ways. First, there is a variable delay between the entrance of a packet into the DLC module at one end of the link and its exit from the other end. This variability is due both to the need to correct the errors that occur at the physical layer and to the variable length of the packets. Second, the time between subsequent entries of packets into the DLC module at one end of the link is also variable. The latter variability is caused both because higher layers might have no packets to send at a given time and also because the DLC is unable to accept new packets when too many old packets are being retransmitted due to transmission errors.

Data link control is discussed in detail in Chapter 2. In essence, the sending DLC module places some overhead control bits called a *header* at the beginning of each packet and some more overhead bits called a *trailer* at the end of each packet, resulting in a longer string of bits called a *frame*. Some of these overhead bits determine if errors have occurred in the transmitted frames, some request retransmissions when errors occur, and some delineate the beginning and ending of frames. The algorithms (or protocols) for accomplishing these tasks are distributed between the peer DLC modules at the two ends of each link and are somewhat complex because the control bits themselves are subject to transmission errors.

The DLC layers in some networks do not retransmit packets in the presence of errors. In these networks, packets in error are simply dropped and retransmission is attempted on an end-to-end basis at the transport layer. The relative merits of this are discussed in Section 2.8.2. Typically, the DLC layer ensures that packets leave the receiving DLC in the same order in which they enter the transmitting DLC, but not all data link control strategies ensure this feature; the relative merits of ordering are also discussed in Section 2.8.2.

Our previous description of the physical layer and DLC was based on point-to-point communication links for which the received waveform at one end of the link is a noisy replica of the signal transmitted at the other end. In some networks, particularly local area networks, some or all of the communication takes place over multiaccess links. For these links, the signal received at one node is a function of the signals from a multiplicity of transmitting nodes, and the signal transmitted from one node might be

heard at a multiplicity of other nodes. This situation arises in satellite communication, radio communication, and communication over cables, optical fibers, and telephone lines with multiple taps. Multiaccess communication is treated in Chapter 4.

The MAC sublayer The appropriate layers for multiaccess communication are somewhat different from those in networks of point-to-point links. There is still the need for a DLC layer to provide a virtual error-free packet link to higher layers, and there is still the need for a physical layer to provide a bit pipe. However, there is also a need for an intermediate layer to manage the multiaccess link so that frames can be sent by each node without constant interference from the other nodes. This is called *medium access control* (MAC). It is usually considered as the lower sublayer of layer 2 with the conventional DLC considered as the higher sublayer. Figure 1.10 illustrates the relationship between these layers. The service provided by the MAC to the DLC is that of an intermittent synchronous bit pipe. The function of the MAC sublayer is to allocate the multiaccess channel so that each node can successfully transmit its frames without undue interference from the other nodes; see Chapter 4 for various ways of accomplishing this function.

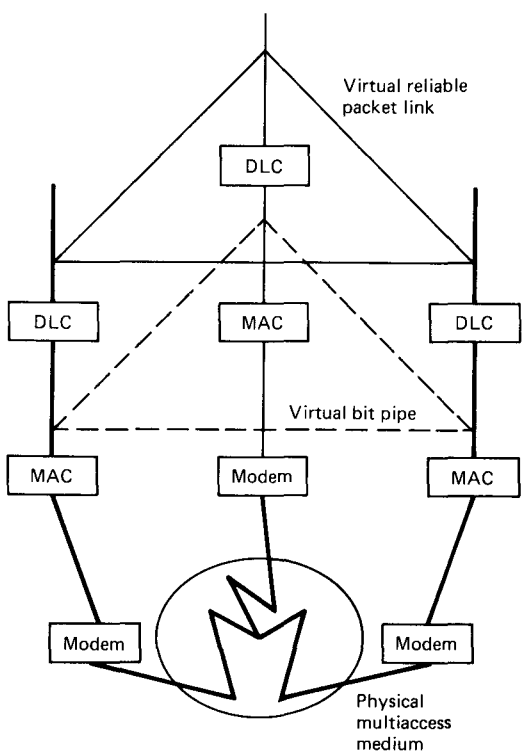


Figure 1.10 Layering for a multiaccess channel. The physical medium is accessed by all three users, each of whom hears the transmitted signals of the others. The DLC sublayer sees virtual point-to-point bit pipes below it. The MAC sublayer sees a multiaccess bit pipe, and the modems access the actual channel.

1.3.3 The Network Layer

The third layer in Fig. 1.8 is the *network layer*. There is one network layer process associated with each node and with each external site of the network. All these processes are peer processes and all work together in implementing routing and flow control for the network. When a frame enters a node or site from a communication link, the bits in that frame pass through the physical layer to the DLC layer. The DLC layer determines where the frame begins and ends, and if the frame is accepted as correct, the DLC strips off the DLC header and trailer from the frame and passes the resulting packet up to the network layer module (see Fig. 1.11). A packet consists of two parts, a packet header followed by the packet body (and thus a frame at the DLC layer contains first the DLC header, next the packet header, next the packet body, and then the DLC trailer). The network layer module uses the packet header of an incoming packet, along with stored information at the module, to accomplish its routing and flow control functions. Part of the principle of layering is that the DLC layer does not look at the packet header or packet body in performing its service function, which is to deliver the packet reliably to the network layer at the next node. Similarly, the network layer does not use any of the information in the DLC header or trailer in performing its functions of routing and flow control. The reason for this separation is to allow improvements, modifications, and replacements in the internal operation of one layer without forcing the other to be modified.

Newly generated messages from users at an external site are processed by the higher layers, broken into packet-sized pieces if need be, and passed down from the transport layer module to the network module. These packet-sized pieces constitute the packet body at the network layer. The transport layer also provides additional information about how to handle the packet (such as where the packet is supposed to go), but this information is passed to the network layer as a set of parameters in accordance with the interfacing protocol between transport and network layer. The network layer module uses

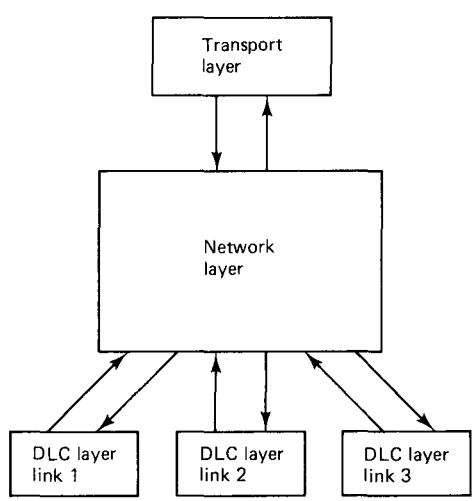


Figure 1.11 The network layer at a node or site can receive packets from a DLC layer for each incoming link and (in the case of a site) from the transport layer. It can send these packets out to the same set of modules.

these parameters, along with its own stored information, to generate the packet header in accordance with the protocol between peer network layer modules.

Along with the transit packets arriving at the network layer module from the lower layer, and the new packets arriving from the higher layer, the network layer can generate its own control packets. These control packets might be specific to a given session, dealing with initiating or tearing down the session, or might have a global function, informing other nodes of link congestion, link failures, and so on.

For networks using virtual circuit routing (*i.e.*, in which the route for a session is fixed over the life of the session), the routing function at the network layer module consists of two parts. The first is to select a route when the virtual circuit is being initiated, and the second is to ensure that each packet of the session follows the assigned route. The selection of a route could be carried out in a distributed way by all the nodes, or could be carried out by the source node or by some central node entrusted with this task. No matter how the job is allocated between the different nodes, however, there is need for considerable communication, via network control packets, concerning the operating characteristics and level of traffic and delay throughout the network. This subject is treated in considerable depth in Chapter 5. Ensuring that each packet follows the assigned route is accomplished by placing enough information in the packet header for the network layer module at each node of the route to be able to forward the packet on the correct outgoing link (or to pass the packet body up to the transport layer when the destination is reached). Ways of doing this are discussed in Section 2.8.

Datagram networks, on the other hand, do not have a connection phase in which a route for a session is determined. Rather, each packet is routed individually. This appears to be a very natural and simple approach, but as Chapter 5 shows, the dynamics of the traffic patterns in a network and the lack of timely knowledge about those patterns at the nodes make this much more difficult than one would think. Most wide area networks use virtual circuits for this reason.

It is necessary here to make a distinction between virtual circuit or datagram *operation* at the network layer and virtual circuit or datagram *service*. The discussion above concerned the *operation* of the network layer; the user of the network layer (usually the transport layer) is concerned only with the *service* offered. Since successive packets of a session, using datagram operation, might travel on different routes, they might appear at the destination out of order. Thus (assuming that the network layer module at the destination does not reorder the packets), the service offered by such a network layer allows packets to get out of order. Typically, with datagram operation, packets are sometimes dropped also. As a result, datagram *service* is usually taken to mean that the network layer can deliver packets out of order, can occasionally fail to deliver packets, and requires no connection phase at the initiation of a session. Conversely, virtual circuit *service* is taken to mean that all packets are delivered once, only once, and in order, but that a connection phase is required on session initiation. We will often use the term *connectionless service* in place of *datagram service* and *connection-oriented service* in place of *virtual circuit service*. We shall see that the difference between connectionless and connection-oriented service has as much to do with quality of service, flow control, and error recovery as it does with routing.

The other major function of the network layer, along with routing, is flow control, or congestion control. Some authors make a distinction between flow control and congestion control, viewing the first as avoiding sending data faster than the final destination can absorb it, and the second as avoiding congestion within the subnet. Actually, if the destination cannot absorb packets as fast as they are sent, those packets will remain in the subnet and cause congestion there. Similarly, if a link in the subnet is congested (*i.e.*, many packets are buffered in an adjacent node waiting for transmission on the link), then there are a number of mechanisms that cause the congestion to spread. Thus congestion is a global issue that involves both the subnet and the external sites, and at least at a conceptual level, it is preferable to treat it as a single problem.

Fundamentally, congestion occurs when the users of the network collectively demand more resources than the network (including the destination sites) has to offer. Good routing can help to alleviate this problem by spreading the sessions out over the available subnet resources. Good buffer management at the nodes can also help. Ultimately, however, the network layer must be able to control the flow of packets into the network, and this is what is meant by flow control (and why we use the term *flow control* in place of *congestion control*).

The control of packet flow into the network must be done in such a way as to prevent congestion and also to provide equitable service to the various sessions. Note that with connection-oriented service, it is possible for a session to negotiate its requirements from the network as a compromise between user desires and network utilization. Thus in some sense the network can guarantee the service as negotiated. With connectionless service, there is no such opportunity for negotiation, and equitable service between users does not have much meaning. This is another reason for the prevalence of connection-oriented service in wide area networks. In Chapter 6 we develop various distributed algorithms for performing the flow control function. As with routing, flow control requires considerable exchange of information between the nodes. Some of this exchange occurs through the packet headers, and some through control packets.

One might hope that the high link capacities that will be available in the future will make it possible to operate networks economically with low utilization, thus making flow control unnecessary. Unfortunately, this view appears overly simplistic. As link capacities increase, access rates into networks will also increase. Thus, even if the aggregate requirements for network service are small relative to the available capacity, a single malfunctioning user could dump enough data into the network quickly to cause serious congestion; if the network plays no regulatory role, this could easily lead to very chaotic service for other users.

The discussion of routing and flow control above has been oriented primarily toward wide area networks. Most local area networks can be viewed as using a single multiaccess channel for communication, and consequently any node is capable of receiving any packet. Thus routing is not a major problem for local area networks. There is a possibility of congestion in local area networks, but this must be dealt with in the MAC sublayer. Thus, in a sense, the major functions of the network layer are accomplished in the MAC sublayer, and the network layer is not of great importance in local area networks. For

this reason, the arguments for virtual circuit operation and connection oriented service in the network layer do not apply to local area networks, and connectionless service is common there.

The network layer is conceptually the most complex of the layered hierarchy since *all* the peer processes at this layer must work together. For the lower layers (except for the MAC sublayer for multiaccess), the peer processes are paired, one at each side of a communication link. For the higher layers, the peer processes are again paired, one at each end of a session. Thus, the network layer and the MAC sublayer are the only layers in which the overall algorithms are distributed between many geographically separated processes.

Acquiring the ability to design and understand such distributed algorithms is one of the basic objectives of this book. Chapter 2 covers the simpler forms of distributed algorithms involving just two peer processes at opposite ends of a link. In Chapter 4 we treat distributed algorithms involving many peer processes in the context of the MAC sublayer, and Chapters 5 and 6 deal with distributed algorithms involving many peer processes at the network layer.

When the network layer and lower layers at all nodes and sites are regarded as one black box, a packet entering the network layer from the next higher layer at a site reappears at some later time at the interface between the network layer and the next higher layer at the destination site. Thus, the network layer appears as a virtual, packet-carrying, end-to-end link from origin site to destination site. Depending on the design of the network layer, this virtual link might be reliable, delivering every packet, once and only once, without errors, or might be unreliable, failing to deliver some packets and delivering some packets with errors. The higher layers then might have to recover from these errors. The network layer might also deliver all packets for each session in order or might deliver them out of order. The relative merits of these alternatives are discussed further in Section 2.8.

The internet sublayer Despite all efforts at standardization, different networks use different algorithms for routing and flow control at the network layer. We have seen some of the reasons for this variety in our discussion of wide area versus local area networks. Since these network layer algorithms are distributed and require close coordination between the various nodes, it is not surprising that one cannot simply connect different subnetworks together. The accepted solution to this problem is to create a new sublayer called the *internet sublayer*. This is usually regarded as being the top part of the network layer. Several subnets can be combined by creating special nodes called gateways between them. A gateway connecting two subnets will interface with each subnet through a network layer module appropriate for that subnet. From the standpoint of the subnet, then, a gateway looks like an external site.

Each gateway will have an internet sublayer module sitting on top of the network layer modules for the individual subnets. When a packet arrives at a gateway from one subnet, the corresponding network layer module passes the packet body and subsidiary information about the packet to the internet module (which thus acts like a transport layer module to the network layer module). This packet body and subsidiary information is

then passed down to the other network layer module for forwarding on through the other subnet.

The internet modules also must play a role in routing and flow control. There is not a great deal of understanding in the field yet as to the appropriate ways for the internet sublayer and the various network layers to work together on routing and flow control. From a practical standpoint, the problem is exacerbated by the fact that the network layers for the subnets are usually in place, designed without the intention of later being used in a network of networks. Thus the internet layer must of necessity be somewhat ad hoc.

When combining local area networks, where routing and flow control are exercised at the MAC sublayer, it is often possible to replace the gateway between subnets with a bridge. Bridges interface different subnets at the DLC layer rather than at the network layer; for local area networks, this is possible because the routing and flow control are done in the MAC sublayer. In Chapter 5 we discuss gateways and bridges in greater detail, particularly with respect to routing.

1.3.4 The Transport Layer

The fourth layer in Fig. 1.8 is the *transport layer*. Here, for each virtual end-to-end link provided by the network layer (or internet sublayer), there is a pair of peer processes, one at each end of the virtual end-to-end link. The transport layer has a number of functions, not all of which are necessarily required in any given network.

First, the transport layer breaks messages into packets at the transmitting end and reassembles packets into messages at the receiving end. This reassembly function is relatively simple if the transport layer process has plenty of buffer space available, but can be quite tricky if there is limited buffer space that must be shared between many virtual end-to-end links. If the network layer delivers packets out of order, this reassembly problem becomes even more difficult.

Second, the transport layer might multiplex several low-rate sessions, all from the same source site and all going to the same destination site, into one session at the network layer. Since the subnet sees only one session in this case, the number of sessions in the subnet and the attendant overhead is reduced. Often this is carried to the extreme in which all sessions with a common source site and common destination site are multiplexed into the same session. In this case, the addressing at the network layer need only specify the source and destination sites; the process within the source site and destination site are then specified in a transport layer header.

Third, the transport layer might split one high-rate session into multiple sessions at the network layer. This might be desirable if the flow control at the network layer is incapable of providing higher-rate service to some sessions than others, but clearly a better solution to this problem would be for the network layer to adjust the rate to the session requirement.

Fourth, if the network layer is unreliable, the transport layer might be required to achieve reliable end-to-end communication for those sessions requiring it. Even when the network layer is designed to provide reliable communication, the transport layer has to be

involved when one or the other end site fails or when the network becomes disconnected due to communication link failures. These failure issues are discussed further in Section 2.8 and in Chapters 5 and 6.

Fifth, end-to-end flow control is often done at the transport layer. There is little difference between end-to-end flow control at the transport layer and network layer (or internet sublayer if it exists). End-to-end flow control at the transport layer is common in practice but makes an integrated approach to avoiding congestion somewhat difficult. This is discussed further in Section 2.9.4 and in Chapter 6.

A header is usually required at the transport layer; this transport header, combined with the data being transported, serves as the packet body passed on to the network layer. Thus the actual body of data is encapsulated in a sequence of headers with the lowest layers on the outside (see Fig. 1.12). At the destination, these layer headers are peeled off in passing up through the various layers. In ISO terminology, the body of data shown in the figure is referred to as a transport service data unit (T-SDU). This data unit, along with the transport header, is referred to as a transport protocol data unit (T-PDU). This unit is also the body of the packet at the network layer, which is sometimes referred to as a network service data unit (N-SDU). Similarly, the packet body plus packet header is referred to as a network protocol data unit (N-PDU). Similarly, each layer in the hierarchy has an SDU, as the unit coming in from the higher layer, and a PDU as the unit going down to the next-lower layer. It is difficult to know where to take a stand against acronymitis in the network field, but we will continue to use the more descriptive terminology of messages, packets, and frames.

1.3.5 The Session Layer

The *session layer* is the next layer above the transport layer in the OSI hierarchy of Fig. 1.8. One function of the session layer is akin to the directory assistance service in the telephone network. That is, if a user wants an available service in the network

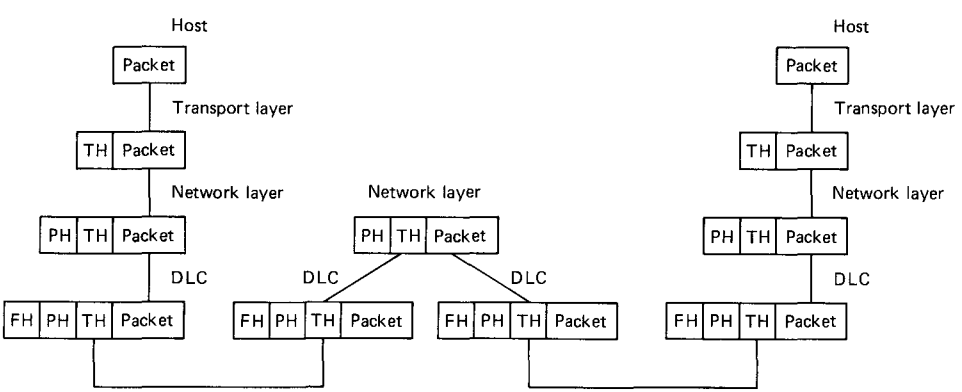


Figure 1.12 Illustration of various headers on a frame. Note that each layer looks only at its own header.

but does not know where to access that service, this layer provides the transport layer with the information needed to establish the session. For example, this layer would be an appropriate place to achieve load sharing between many processors that are sharing computational tasks within a network.

The session layer also deals with *access rights* in setting up sessions. For example, if a corporation uses a public network to exchange records between branch offices, those records should not be accessible to unauthorized users. Similarly, when a user accesses a service, the session layer helps deal with the question of who pays for the service.

In essence, the session layer handles the interactions between the two end points in setting up a session, whereas the network layer handles the subnet aspects of setting up a session. The way that session initiation is divided between session layer, transport layer, and network layer varies from network to network, and many networks do not have these three layers as separate entities.

1.3.6 The Presentation Layer

The major functions of the *presentation layer* are data encryption, data compression, and code conversion. The need for encryption in military organizations is obvious, but in addition, corporations and individual users often must send messages that should only be read by the intended recipient. Although data networks should be designed to prevent messages from getting to the wrong recipients, one must expect occasional malfunctions both at the external sites and within the subnet; this leads to the need for encryption of critical messages.

The desirability of data compression in reducing the number of bits to be communicated has already been mentioned. This function could be performed at any of the layers, but there is some advantage in compressing the data for each session separately, in the sense that different sessions have different types of redundancy in their messages. In particular, data compression must be done (if at all) before encryption, since encrypted data will not have any easily detectable redundancy.

Finally, code conversion is sometimes necessary because of incompatible terminals, printers, graphics terminals, file systems, and so on. For example, some terminals use the ASCII code to represent characters as 8-bit bytes, whereas other terminals use the EBCDIC code. Messages using one code must be converted to the other code to be readable by a terminal using the other code.

1.3.7 The Application Layer

The *application layer* is simply what is left over after the other layers have performed their functions. Each application requires its own software (*i.e.*, peer processes) to perform the desired application. The lower layers perform those parts of the overall task that are required for many different applications, while the application layer does that part of the task specific to the particular application.

At this point, the merits of a layered approach should be clear, but there is some question about which functions should be performed at each layer. Many networks omit

the session and presentation layers, and as we have seen, the lower layers are now divided into sublayers. An even more serious issue is that in an effort to achieve agreement on the standards, a number of alternatives have been introduced which allow major functions to be either performed or not at various layers. For example, error recovery is sometimes done at the DLC layer and sometimes not, and because of this, the higher layers cannot necessarily count on reliable packet transfer. Thus, even within the class of networks that conform to the OSI reference model, there is considerable variation in the services offered by the various layers. Many of the existing networks described later do not conform to the OSI reference model, and thus introduce even more variation in layer services. Broadband ISDN networks, for example, do routing and flow control at the physical layer (in a desire to simplify switch design), thus making the network look like an end-to-end bit pipe from the origin to destination.

Even with all these problems, there is a strong desire for standardization of the interfaces and functions provided by each layer, even if the standard is slightly inappropriate. This desire is particularly strong for international networks and particularly strong among smaller equipment manufacturers who must design equipment to operate correctly with other manufacturers' equipment. On the other hand, standardization is an impediment to innovation, and this impediment is particularly important in a new field, such as data networks, that is not yet well understood. Fortunately, there is a constant evolution of network standards. For example, the asynchronous transfer node (ATM) protocol of broadband ISDN circumvents the ISO network layer standard by performing the function of the network layer at the physical layer (see Section 2.10).

One particular difficulty with the seven layers is that each message must pass through seven processes before even entering the subnet, and all of this might generate considerable delay. This text neither argues for this particular set of layers nor proposes another set. Rather, the objective is to create an increased level of understanding of the functions that must be accomplished, with the hope that this will contribute to standardization. The existing networks examined in subsequent chapters do not, in fact, have layers that quite correspond to the OSI layers.

1.4 A SIMPLE DISTRIBUTED ALGORITHM PROBLEM

All of the layers discussed in Section 1.3 have much in common. All contain peer processes, one at each of two or more geographically separated points, and the peer processes communicate via the communication facility provided by the next lower layer. The peer processes in a given layer have a common objective (*i.e.*, task or function) to be performed jointly, and that objective is achieved by some combination of processing and interchanging information. The algorithm to achieve the objective is a *distributed algorithm* or a *protocol*. The distributed algorithm is broken down into a set of *local algorithms*, one of which is performed by each peer process. The local algorithm performed by one process in a set of peers consists of carrying out various operations on the available data, and at various points in the algorithm, either sending data to one or more other peer processes or reading (or waiting for) data sent by another peer process.

In the simplest distributed algorithm, the order in which operations are carried out by the various local algorithms is completely determined. For example, one local algorithm might perform several operations and then reliably send some data to the other local algorithm, which then carries out some operations and returns some data. Only one local algorithm operates at a time and the distributed algorithm is similar to a centralized algorithm that performs all operations sequentially at one location.

In more complex cases, several local algorithms might operate concurrently, but each still waits at predetermined points in the local algorithm for predetermined messages from specific other local algorithms. In this case, the overall distributed algorithm still operates in a deterministic fashion (given the input data to the peer processes), but the lockstep ordering of operations between different local algorithms is removed.

In the most complex case (which is of most interest), the order in which a local algorithm performs its operations depends on the order in which data arrive (either from the next higher layer or from a peer process). Also, if the underlying communication facility is unreliable, data sent by a peer process might never arrive or might arrive with errors.

Most people are very familiar with the situation above, since people must often perform tasks requiring interacting with others, often with unreliable communication. In these situations, however, people deal with problems as they arise rather than thinking through all possible eventualities ahead of time, as must be done with a distributed algorithm.

To gain some familiarity with distributed algorithms, a very simple problem is presented, involving unreliable communication, which in fact has no solution. Analogous situations arise frequently in the study of data networks, so it is well to understand such a problem in its simplest context.

There are three armies, two colored magenta and one lavender. The lavender army separates the two magenta armies, and if the magenta armies can attack simultaneously, they win, but if they attack separately, the lavender army wins. The only communication between the magenta armies is by sending messengers through the lavender army lines, but there is a possibility that any such messenger will be captured, causing the message to go undelivered (see Fig. 1.13). The magenta armies would like to synchronize their attack at some given time, but each is unwilling to attack unless assured with certainty that the other will also attack. Thus, the first army might send a message saying, "Let's attack on Saturday at noon; please acknowledge if you agree."

The second army, hearing such a message, might send a return message saying, "We agree; send an acknowledgment if you receive our message." It is not hard to see that this strategy leads to an infinite sequence of messages, with the last army to send a message being unwilling to attack until obtaining a commitment from the other side.

What is more surprising is that no strategy exists for allowing the two armies to synchronize. To see this, assume that each army is initially in state 0 and stays in this state if it receives no messages. If an army commits itself to attack, it goes to state 1, but it will not go to state 1 unless it is certain that the other army will go to state 1. We also assume that an army can change state only at the time that it receives a message (this assumption in essence prevents side information other than messages from synchronizing the armies). Now consider any ordering in which the two armies receive messages. The

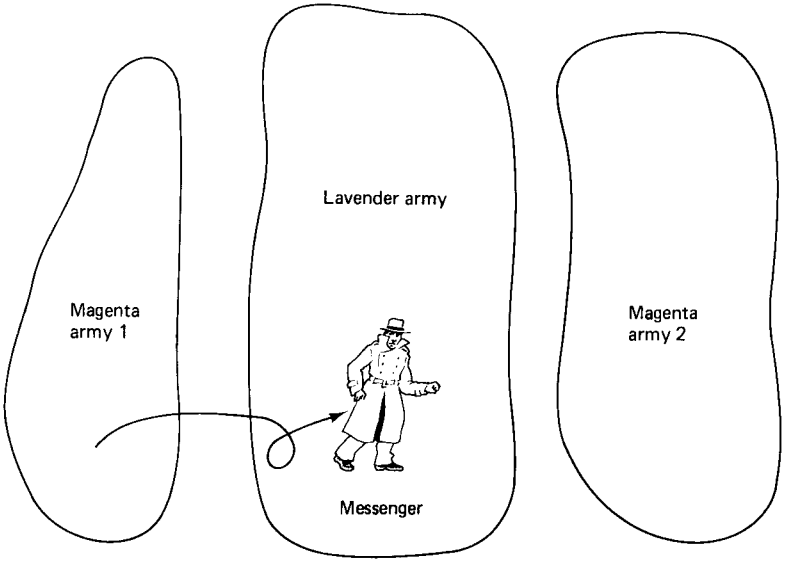


Figure 1.13 A messenger carries a message through enemy lines from magenta army 1 to magenta army 2. If the messenger is caught, the message is undelivered. Magenta army 1 is unaware of capture and magenta army 2 is unaware of existence of message.

first army to receive a message cannot go to state 1, since it has no assurance that any message will be received by the other army. The second army to receive a message also cannot go to state 1 since it is not assured that the other side will receive subsequent messages, and even if it knows that the other side received a first message, it knows that the other side is not currently in state 1. Proceeding in this way (or more formally by induction), neither army can ever go to state 1.

What is surprising about this argument is the difficulty in convincing oneself that it is correct. The difficulty does not lie with the induction argument, but rather with the question of whether the model fairly represents the situation described. It appears that the problem is that we are not used to dealing with distributed questions in a precise way; classical engineering problems do not deal with situations in which distributed decisions based on distributed information must be made.

If the conditions above are relaxed somewhat so as to require only a high probability of simultaneous attack, the problem can be solved. The first army simply decides to attack at a certain time and sends many messengers simultaneously to the other side. The first army is then assured with high probability that the second army will get the message, and the second army is assured that the first army will attack.

Fortunately, most of the problems of communication between peer processes that are experienced in data networks do not require this simultaneous agreement. Typically, what is required is for one process to enter a given state with the assurance that the peer process will *eventually* enter a corresponding state. The first process might be required to wait for a confirmation of this eventuality, but the deadlock situation of the three-army problem, in which neither process can act until after the other has acted, is avoided.

NOTES AND SUGGESTED READING

The introductory textbooks by Tanenbaum [Tan88], Stallings [Sta85], and Schwartz [Sch87] provide alternative treatments of the material in this chapter. Tanenbaum's text is highly readable and contains several chapters on the higher levels of the OSI architecture. Stalling's text contains a wealth of practical detail on current network practice. Schwartz's text also includes several chapters on circuit switching. Some perspectives on the historical evolution of data networks are given in [Gre84].

New developments in technology and applications are critically important in both network design and use. There are frequent articles in the *IEEE Spectrum*, *IEEE Communications Magazine*, and *IEEE Computer* that monitor these new developments. *Silicon Dreams: Information, Man and Machine* by Lucky [Luc90] provides an excellent overview of these areas. A good reference on layered architecture is [Gre82], and some interesting commentary on future standardization of layers is given in [Gre86].

PROBLEMS

1.1. A high quality image requires a spatial resolution of about 0.002 inch, which means that about 500 pixels (*i.e.* samples) per inch are needed in a digital representation. Assuming 24 bits per pixel for a color image of size 8.5 by 11 inches, find the total number of bits required for such an image representation.

1.2. **(a)** Suppose a city of one million inhabitants builds a data network. Suppose that each inhabitant, during the busiest hour of the day, is involved in an average of 4 transactions per hour that use this network (such as withdrawing money from a cash machine, buying some item in a store and thus generating an inventory control message, etc.). Suppose that each transaction, on the average, causes 4 packets of 1000 bits each to be sent. What is the aggregate average number of bits per second carried by the network? How many 64 kbit/sec voice telephone links are required to carry this traffic assuming that each packet travels over an average of 3 links?

(b) Suppose that the inhabitants use their telephones an average of 10% of the time during the busy hour. How many voice telephone links are required for this, assuming that all calls are within the city and travel over an average of three links?

1.3. Suppose packets can get dropped or arbitrarily delayed inside a packet network. Suppose two users are communicating in a session and want to terminate the session. We would like a protocol that exchanges packets in such a way that both users know that they can terminate with the knowledge that no further packets will arrive from the other user. Can such a protocol be designed? What is the relation between this problem and the three-army problem of Section 1.10?