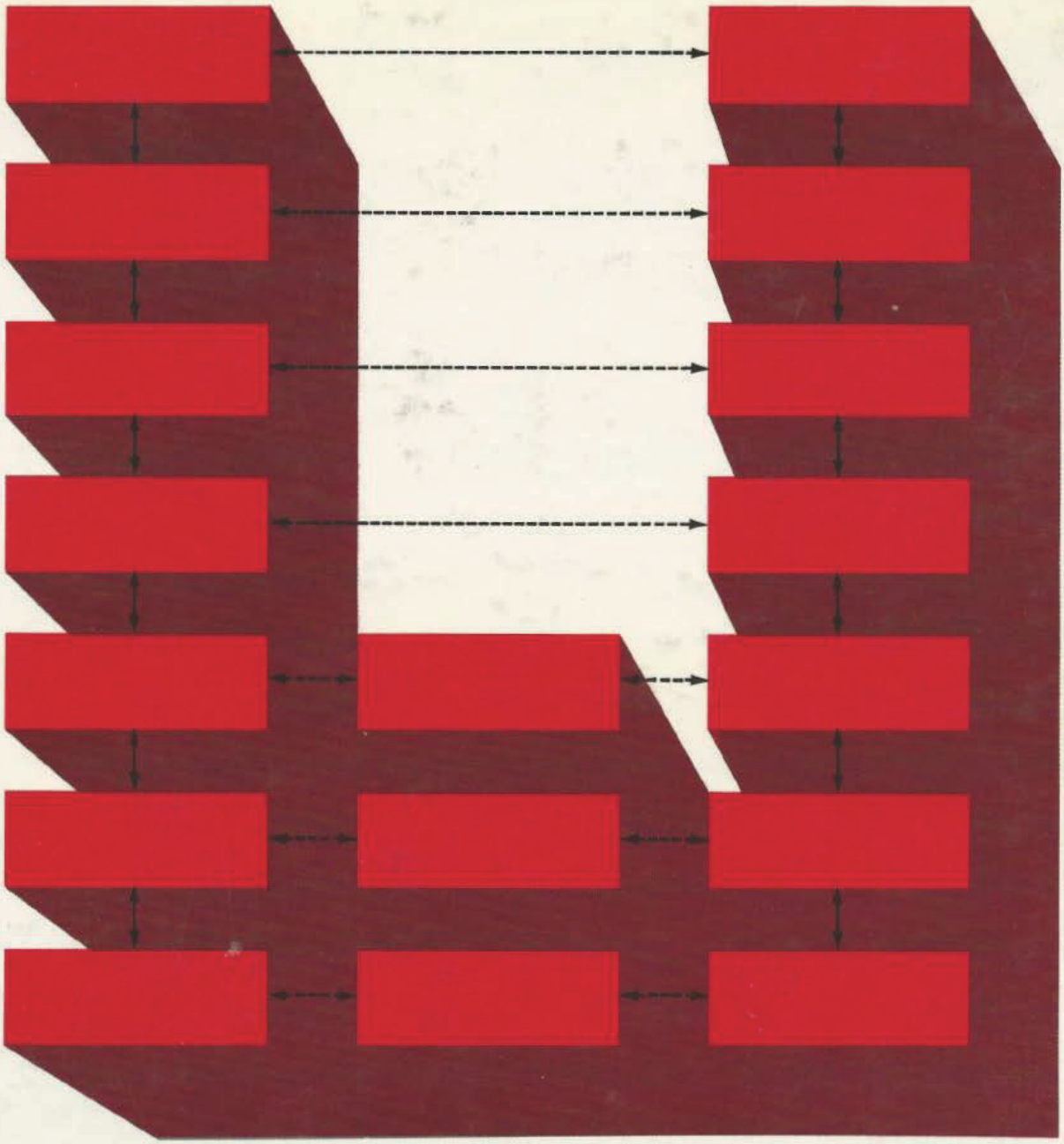
SECOND EDITION

COMPUTER NETWORKS



ANDREW S. TANENBAUM

APPLE-1019

Library of Congress Cataloging-in-Publication Data
Tanenbaum, Andrew S.
Computer networks / Andrew S. Tanenbaum.—2nd ed.
p. cm.
Bibliography: p.
Includes index.
ISBN 0-13-162959-X
1. Computer networks. I. Title.
TK5105.5.T36 1988
004.6—dc19 88-16937
CIP

Editorial/production supervision: **Lisa Schulz Garboski**
Manufacturing buyer: **Mary Ann Gloriande**

Prentice Hall Software Series
Brian W. Kernighan, Advisor

The publisher offers discounts on this book when ordered in bulk quantities. For more information, write:

Special Sales/College Marketing
Prentice Hall
College Technical and Reference Division
Englewood Cliffs, NJ 07632



© 1988 by Prentice-Hall, Inc.
A Division of Simon & Schuster
Englewood Cliffs, New Jersey 07632

All rights reserved. No part of this book may be reproduced, in any form or by any means, without permission in writing from the publisher.

Printed in the United States of America

10 9 8 7 6 5 4 3

ISBN 0-13-162959-X

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

computer network as a sophisticated communication system may reduce the amount of traveling done, thus saving energy. Home work may become popular, especially for part-time workers with young children. The office and school as we now know them may disappear. Stores may be replaced by electronic mail order catalogs. Cities may disperse, since high-quality communication facilities tend to reduce the need for physical proximity. The information revolution may change society as much as the Industrial Revolution did.

1.2. NETWORK STRUCTURE

It is now time to turn our attention from the social implications of networking to the technical issues involved in network design. In any network there exists a collection of machines intended for running user (i.e., application) programs. We will follow the terminology of one of the first major networks, the ARPANET, and call these machines **hosts**. The term **end system** is sometimes also used in the literature. The hosts are connected by the **communication subnet**, or just **subnet** for short. The job of the subnet is to carry messages from host to host, just as the telephone system carries words from speaker to listener. By separating the pure communication aspects of the network (the subnet) from the application aspects (the hosts), the complete network design is greatly simplified.

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

The switching elements are specialized computers used to connect two or more transmission lines. When data arrive on an incoming line, the switching element must choose an outgoing line to forward them on. Again following the original ARPANET terminology, we will call the switching elements **IMPs** (**Interface Message Processors**) throughout the book, although the terms **packet switch node**, **intermediate system**, and **data switching exchange** are also commonly used. Unfortunately, there is no consensus on terminology here; every writer on the subject seems to use a different name. The term "IMP" is probably as good as any. In this model, shown in Fig. 1-2, each host is connected to one (or occasionally several) IMPs. All traffic to or from the host goes via its IMP.

Broadly speaking, there are two types of designs for the communication subnet:

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

In the first one, the network contains numerous cables or leased telephone lines, each one connecting a pair of IMPs. If two IMPs that do not share a cable nevertheless wish to communicate, they must do this indirectly, via other IMPs.

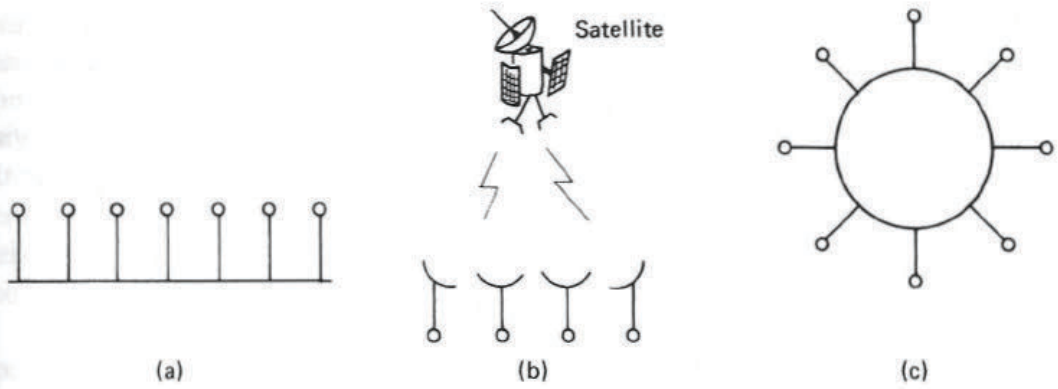


Fig. 1-4. Communication subnets using broadcasting. (a) Bus. (b) Satellite or radio. (c) Ring.

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

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

1.3. NETWORK ARCHITECTURES

Modern computer networks are designed in a highly structured way. In the following sections we examine the structuring technique in some detail.

1.3.1. Protocol Hierarchies

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

Layer n on one machine carries on a conversation with layer n on another machine. The rules and conventions used in this conversation are collectively known as the layer n **protocol**, as illustrated in Fig. 1-5 for a seven-layer network. The entities comprising the corresponding layers on different machines are called **peer processes**. In other words, it is the peer processes that communicate using the protocol.

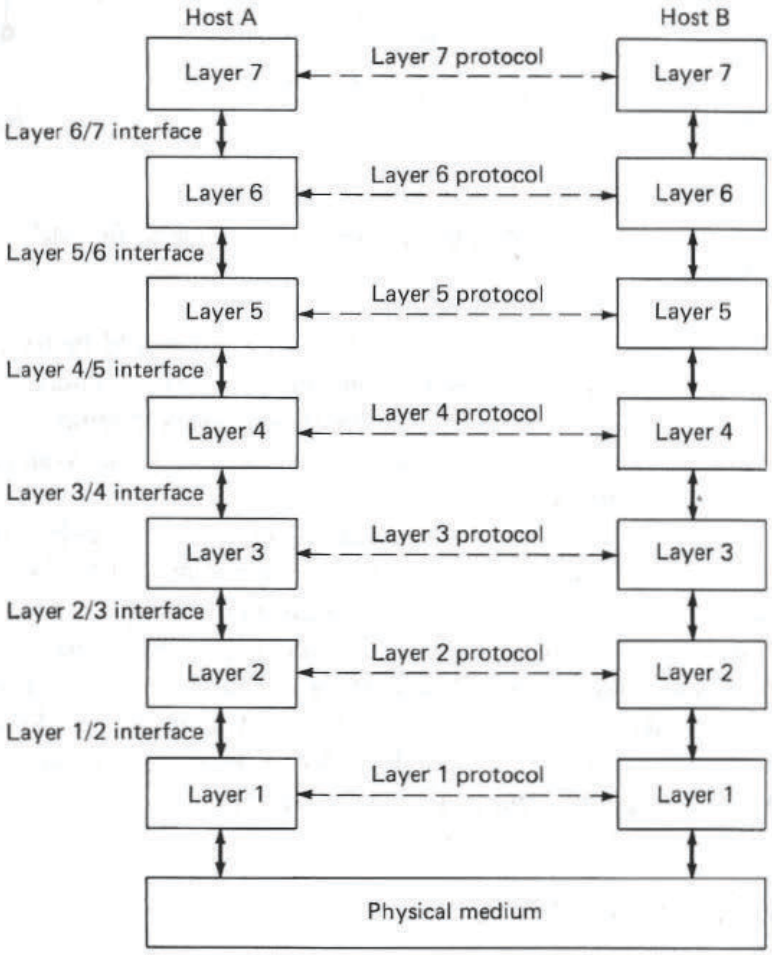


Fig. 1-5. Layers, protocols, and interfaces.

In reality, no data are directly transferred from layer n on one machine to layer n on another machine. Instead, each layer passes data and control information to the layer immediately below it, until the lowest layer is reached. Below layer 1 is the **physical medium** through which actual communication occurs. In Fig. 1-5, virtual communication is shown by dotted lines and physical communication by solid lines.

Between each pair of adjacent layers there is an **interface**. The interface defines which primitive operations and services the lower layer offers to the upper one. When network designers decide how many layers to include in a network and

demultiplexing is done transparently, it can be used by any layer. Multiplexing is needed in the physical layer, for example, where all the traffic for all connections has to be sent over at most a few physical circuits.

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

1.4. THE OSI REFERENCE MODEL

Now that we have discussed layered networks in the abstract, it is time to look at the set of layers that will be used throughout this book. The model is shown in Fig. 1-7. This model is based on a proposal developed by the International Standards Organization (ISO) as a first step toward international standardization of the various protocols (Day and Zimmermann, 1983). The model is called the **ISO OSI Open Systems Interconnection) Reference Model** because it deals with connecting open systems—that is, systems that are open for communication with other systems. We will usually just call it the OSI model for short.

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

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

In Sections 1.5.1 through 1.5.7 we will discuss each layer of the model in turn, starting at the bottom layer. Note that the OSI model itself is not a network architecture because it does not specify the exact services and protocols to be used in each layer. It just tells what each layer should do. However, ISO has also produced standards for all the layers, although these are not strictly speaking part of the model. Each one has been published as a separate international standard.

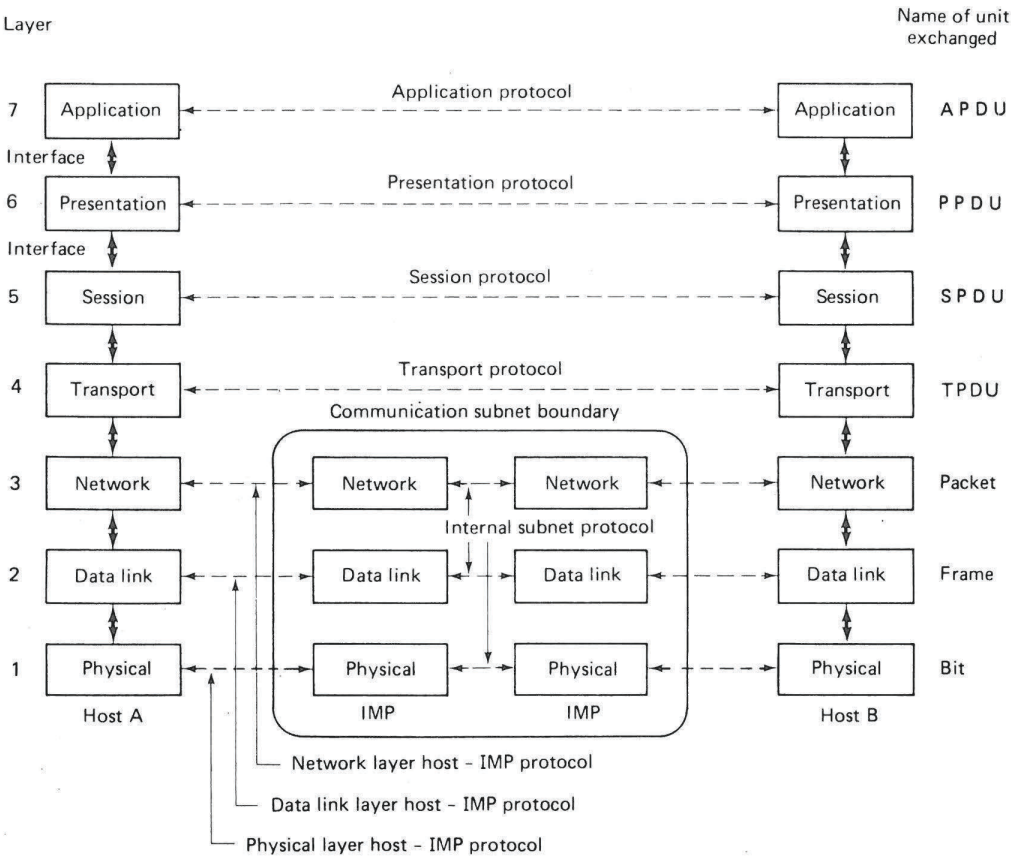


Fig. 1-7. The network architecture used in this book. It is based on the OSI model.

1.4.1. The Physical Layer

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

1.4.2. The Data Link Layer

The main task of the **data link layer** is to take a raw transmission facility and transform it into a line that appears free of transmission errors to the network layer. It accomplishes this task by having the sender break the input data up into **data frames** (typically a few hundred bytes), transmit the frames sequentially, and process the **acknowledgement frames** sent back by the receiver. Since the physical layer merely accepts and transmits a stream of bits without any regard to meaning or structure, it is up to the data link layer to create and recognize frame boundaries. This can be accomplished by attaching special bit patterns to the beginning and end of the frame. If these bit patterns can accidentally occur in the data, special care must be taken to avoid confusion.

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

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

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

1.4.3. The Network Layer

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

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

Since the operators of the subnet may well expect remuneration for their efforts,

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

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

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

1.4.4. The Transport Layer

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

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

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

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

Many hosts are multiprogrammed, which implies that multiple connections will be entering and leaving each host. There needs to be some way to tell which message belongs to which connection. The transport header (H_4 in Fig. 1-6) is one place this information could be put.

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

1.4.5. The Session Layer

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

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

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

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

1.4.6. The Presentation Layer

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

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

The presentation layer is also concerned with other aspects of information representation. For example, data compression can be used here to reduce the number of bits that have to be transmitted and cryptography is frequently required for privacy and authentication.

1.4.7. The Application Layer

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

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

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

1.4.8. Data Transmission in the OSI Model

Figure 1-8 shows an example of how data can be transmitted using the OSI model. The sending process has some data it wants to send to the receiving process. It gives the data to the application layer, which then attaches the application

header, *AH* (which may be null), to the front of it and gives the resulting item to the presentation layer.

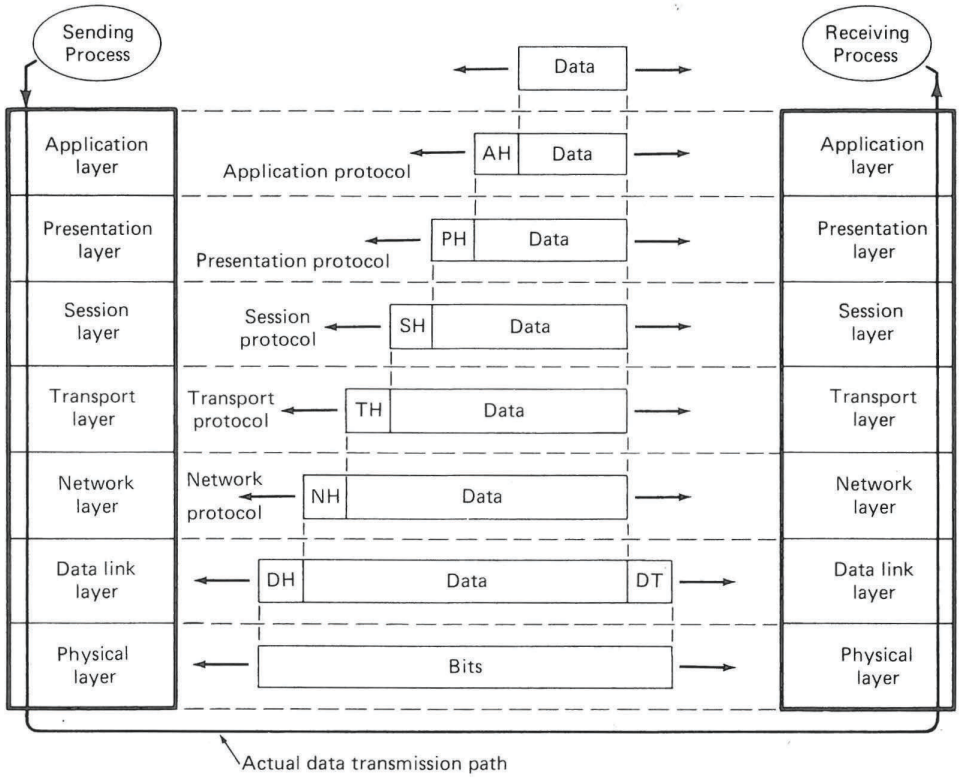


Fig. 1-8. An example of how the OSI model is used. Some of the headers may be null. (Source: H.C. Folts. Used with permission.)

The presentation layer may transform this item in various ways, and possibly add a header to the front, giving the result to the session layer. It is important to realize that the presentation layer is not aware of which portion of the data given to it by the application layer is *AH*, if any, and which is true user data. Nor should it be aware.

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

The key idea throughout is that although actual data transmission is vertical in Fig. 1-8, each layer is programmed as though it were really horizontal. When the sending transport layer, for example, gets a message from the session layer, it attaches a transport header and sends it to the receiving transport layer. From its

point of view, the fact that it must actually hand the message to the network layer on its own machine is an unimportant technicality. As an analogy, when an Uighur-speaking diplomat is addressing the United Nations, he thinks of himself as addressing the other assembled diplomats. That, in fact, he is really only speaking to his translator is seen as a technical detail.

1.5. SERVICES

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

1.5.1. OSI Terminology

The active elements in each layer are called **entities**. An entity can be a software entity (such as a process), or a hardware entity (such as an intelligent I/O chip). Entities in the same layer on different machines are called **peer entities**. The layer 7 entities are called **application entities**; the layer 6 entities are called **presentation entities**, and so on.

The entities in layer N implement a service used by layer $N + 1$. In this case layer N is called the **service provider** and layer $N + 1$ is called the **service user**. Layer N may use the services of layer $N - 1$ in order to provide its service. It may offer several classes of service, for example, fast, expensive communication and slow, cheap communication.

Services are available at **SAPs**. (**service access points**), The layer N SAPs are the places where layer $N + 1$ can access the services offered. Each SAP has an address that uniquely identifies it. To make this point clearer, the SAPs in the telephone system are the sockets into which modular telephones can be plugged, and the SAP addresses are the telephone numbers of these sockets. To call someone, you must know his SAP address. Similarly, in the postal system, the SAP addresses are street addresses and post office boxes. To send a letter, you must know the addressee's SAP address. In Berkeley UNIX†, the SAPs are the sockets and the SAP addresses are the socket numbers. The SAP concept is discussed in detail by Tomas et al. (1987).

In order for two layers to exchange information, there has to be an agreed upon set of rules about the **interface**. At a typical interface, the layer $N + 1$ entity passes an **IDU (Interface Data Unit)** to the layer N entity through the SAP as shown in Fig. 1-9. The IDU consists of an **SDU (Service Data Unit)** and some control information. The SDU is the information passed across the network to the peer entity

† UNIX is a registered trademark of AT&T Bell Laboratories.

1.7.2. The ARPANET

The ARPANET is the creation of ARPA (now DARPA), the (Defense) Advanced Research Projects Agency of the U.S. Department of Defense. Starting in the late 1960s it began stimulating research on the subject of computer networks by providing grants to computer science departments at many U.S. universities, as well as to a few private corporations. This research led to an experimental four-node network that went on the air in December 1969. It has been operating ever since, and has subsequently grown to several hundred computers spanning half the globe, from Hawaii to Sweden. Much of our present knowledge about networking is a direct result of the ARPANET project. To honor this pioneering work, we have adopted some of the original ARPANET terminology (e.g., host, IMP, subnet).

After the ARPANET technology had proven itself by years of highly reliable service, a military network, MILNET, was set up using the same technology. An extension of MILNET in Europe, called MINET, was also created. MILNET and MINET are connected to the ARPANET, but the traffic between the parts is tightly controlled. Two satellite networks, SATNET and WIDEBAND, were also hooked up later. Since many of the universities and government contractors on the ARPANET had their own LANs, eventually these were also connected to the IMPs, leading to the **ARPA internet** with thousands of hosts and well over 100,000 users. The ARPANET will be frequently cited throughout this book, but most of what is said about the ARPANET (e.g., the protocols used) also holds for the entire internet.

The original ARPANET IMPs were Honeywell DDP-516 minicomputers with 12K 16-bit words of memory. As time has gone on, the IMPs have been replaced several times by more powerful machines. They are now called **PSNs** (**Packet Switch Nodes**), but their function is the same as it was.

Some of the IMPs have been configured to allow user terminals to call them directly, instead of logging into a host. These were called **TIPs** (**Terminal Interface Processors**) and are now called **TACs** (**Terminal Access Controllers**). The IMPs were originally connected by 56 kbps leased lines, although higher speed (e.g., 230.4 kbps), lines are now also used. The original IMPs could handle one to four hosts apiece. The current ones can handle tens of hosts and hundreds of terminals simultaneously.

The ARPANET does not follow the OSI model at all (it predates OSI by more than a decade). The IMP-IMP protocol really corresponds to a mixture of the layer 2 and layer 3 protocols. Layer 3 also contains an elaborate routing mechanism. In addition, there is a mechanism that explicitly verifies the correct reception at the destination IMP of each and every packet sent by the source IMP. Strictly speaking, this mechanism is another layer of protocol, the source IMP to destination IMP protocol. However, this protocol does not exist in the OSI model. We will treat it as part of layer 3, since it is closer to that layer than to any other.

The ARPANET does have protocols that roughly cover the same territory as the

OSI network and transport protocols. The network protocol, called **IP (Internet Protocol)**, is connectionless and was designed to handle the interconnection of the vast number of WAN and LAN networks comprising the ARPA internet. The OSI model only grudgingly dealt with internetworking at all, as an afterthought, whereas the concept was central to the design of IP.

The ARPANET transport protocol is a connection-oriented protocol called **TCP (Transmission Control Protocol)**. It resembles the OSI transport protocol in its general style, but it differs in all the formats and details. TCP is used in Berkeley UNIX, which makes it very widespread even though it is not part of the OSI suite. Just as an aside, it should be pointed out that TCP is the second generation transport protocol. The first one is no longer used.

There are no session or presentation layer protocols in the ARPANET as no one has had any use for them in the first 20 years of operation. Various application protocols exist, but they are not structured the same way as their OSI counterparts. The ARPANET services include file transfer, electronic mail, and remote login. These services are supported by the well-known protocols **FTP (File Transfer Protocol)**, **SMTP (Simple Mail Transfer Protocol)**, and **TELNET** (remote login). Various specialized protocols are available for other applications.

1.7.3. MAP and TOP

If two sites each use the OSI model, there is no guarantee that they will be able to communicate with each other. The model consists of a framework telling what the layers are, but does not itself specify the protocols to be used in each layer. While ISO has standardized certain protocols, other nonstandard protocols also exist. Furthermore, in some layers, multiple, incompatible standard protocols also exist. If two sites use different protocols in any layer, they will not be able to communicate.

Many readers may just throw up their hands in disgust and say: "Why can't the experts just agree on one standard protocol in each layer?" It is instructive to look at a concrete example to see how the problem arises. In 1973, Robert Metcalfe wrote a Ph.D. thesis at M.I.T. in which he described his research on LANs. Metcalfe then moved to the Xerox Corporation where he teamed up with David Boggs and others to implement the **Ethernet**† LAN based on the ideas in Metcalfe's thesis. Ethernet was quickly adopted by many companies and Intel later built a single-chip controller for it. It did not take long before Ethernet became the de facto standard for LANs.

Shortly thereafter, some people felt that there should be an *official* standard for LANs. A group of people working under the auspices of IEEE was set up to write

† Ethernet is a trademark of the Xerox Corporation.

for all departments, not just computer science. It has now spread to about 175 sites in the U.S. and about 260 sites in Europe via its counterpart there, called **EARN** (**European Academic Research Network**).

Technically, BITNET is somewhat similar to CYPRESS, with each BITNET site leasing a line to some other site. Unlike CYPRESS, the BITNET hosts themselves do the communication; there is no subnet of IMPLETs. Also unlike CYPRESS, BITNET uses a protocol and software donated to it by IBM that is not compatible with either OSI or TCP/IP (or anything else). It is based on the idea of transmitting 80-column punched card images, a frequent source of problems.

The financing of BITNET is unusual, and accounts for part of its popularity. To join, a university must lease a line to some other BITNET site and pay the rental cost of the line. It must also permit some other university to lease a line to it in the future (at the other university's expense). Finally, it must agree to forward traffic passing through it for free. Other than the loss of some computing power for forwarding third-party traffic, the only real cost is one leased line. Unlike just about every other network, there are no charges based on the volume of traffic sent.

The basic BITNET service is file transfer, which also includes electronic mail and remote job entry. Each file entered into the system contains its final destination, and may be stored and forwarded many times before it gets there. A limited amount of remote login is possible, but since the interactive traffic is stored and forwarded just like the file transfer traffic, response time is very slow and reliability is low.

Work is in progress to add a CYPRESS-like subnet to BITNET, and to try to integrate it better into the ARPA internet, USENET, and CSNET worlds.

1.7.7. SNA

No discussion of networking would be complete without at least a few words about IBM's network architecture, **SNA** (**Systems Network Architecture**). The OSI model was patterned after SNA to a considerable extent, including the concept of layering, the number of layers chosen, and their approximate functions.

SNA is a network architecture intended to allow IBM customers to construct their own private networks, both the hosts and the subnet. A bank, for example, might have one or more CPUs in its data-processing department and numerous terminals in each of its branch offices. Using SNA, all these isolated components could be transformed into a coherent system.

Prior to SNA, IBM had several hundred communication products, using three dozen teleprocessing access methods, with more than a dozen data link protocols alone. The idea behind SNA was to eliminate this chaos and to provide a coherent framework for loosely coupled distributed processing. Given the desire of many of IBM's customers to maintain compatibility with all these (mutually incompatible) programs and protocols, the SNA architecture is more complicated in places than it might have been had these constraints not been present. SNA also performs a large

number of functions not found in other networks, which, although valuable for certain applications, tend to add to the overall complexity of the architecture.

SNA has evolved considerably over the years and is still evolving. The original release in 1974 permitted only centralized networks, that is, tree-shaped networks with only a single host and its terminals. From our point of view, that is no network at all. The 1976 release allowed multiple hosts with their respective trees, with intertree communication possible only between the roots of the trees. The 1979 release removed this restriction, allowing somewhat more general communication. Finally, in 1985, arbitrary topologies of hosts and LANs were supported.

An SNA network consists of a collection of machines called **nodes**, of which there are four types, approximately characterized as follows. Type 1 nodes are terminals. Type 2 nodes are controllers, machines that supervise the behavior of terminals and other peripherals. Type 4 nodes are front end processors, devices whose function is to relieve the main CPU of the work and interrupt handling associated with data communication. Type 5 nodes are the main hosts themselves, although with the advent of low-cost microprocessors, some controllers have acquired some host-like properties. Strangely enough, there are no type 3 nodes.

Each node contains one or more **NAU**s (**Network Addressable Units**). A NAU is a piece of software that allows a process to use the network. It can be regarded as a SAP plus the entities that provide the upper layer services. To use the network, a process must connect itself to a NAU, at which time it can be addressed and can address other NAUs. The NAUs are thus the entry points into the network for user processes.

There are three kinds of NAUs. An **LU** (**logical unit**) is the usual variety to which user processes can be attached. A **PU** (**physical unit** is a special administrative NAU associated with each node. The PU is used by the network to bring the node online, take it offline, test it, and perform similar network management functions. A PU provides a way for the network to address a physical device, without reference to which processes are using it. The third kind of NAU is the **SSCP** (**System Services Control Point**) of which there is normally one per type 5 node and none in the other nodes. The SSCP has complete knowledge of, and control over, all the front ends, controllers, and terminals attached to the host. The collection of hardware and software managed by an SSCP is called a **domain**. Figure 1-17 depicts a simple two-domain SNA network.

While it is possible to make a rough mapping of the SNA layers to the OSI layers, when you look at them in detail, the two models do not correspond especially well, especially in layers 3, 4, and 5. A summary of the SNA layers follows.

The lowest SNA layer, shown in Fig. 1-18 takes care of physically transporting bits from one machine to another. The protocols used in this layer generally conform to the appropriate industry standards.

The next layer, the **data link control** layer, constructs frames from the raw bit stream, detecting and recovering from transmission errors in a way transparent to higher layers. Many networks have directly or indirectly copied their layer 2

violate the normal queue ordering and skip to the head of the queue. Whether or not they actually do this, is implementation dependent. A typical example of expedited data is a user at a terminal hitting the DEL key to interrupt a running program. The DEL packet will go as expedited data.

The *qos* parameter is actually two lists of values that determine the **quality of service** provided by the connection. The first list gives the goal—what the caller really wants. The second list gives the minimum values considered acceptable. If the network service is unable to provide at least the minimum value for any parameter specified by either the caller (calling party) or the callee (called party), the connection establishment fails. The values that may be specified include throughput, delay, error rate, secrecy, and cost, among others.

The caller may include some user data in the connection request. The callee may inspect these data before deciding whether to accept or reject the request. Connection requests are accepted with the *N-CONNECT.response* primitive and rejected with the *N-DISCONNECT.request* primitive. When a request is rejected, the *reason* field allows the callee to tell why it was not willing to accept the connection and whether the condition is permanent or transient. The network layer itself may also reject attempts to establish connections, for example, if the quality of service desired is not available (permanent condition) or the subnet is currently overloaded (transient condition).

The remaining *N-CONNECT* primitives and the *N-DISCONNECT* primitives are straightforward, and need little further comment. After a connection has been established, either party can transmit data using *N-DATA.request*. When these packets arrive, an *N-DATA.indication* primitive is invoked on the receiving end. Expedited data uses primitives analogous to those for regular data.

If acknowledgements have been agreed upon, when a packet has been received, the recipient is expected to issue an *N-DATA-ACKNOWLEDGE.request*. This primitive contains no sequence number, so the party sending the original data must simply count acknowledgements. If the quality of service is low, and data and acknowledgements can be lost, this scheme is not very satisfactory. On the other hand, it is not the task of the network layer to provide an error-free service; that job is done by the transport layer. Network layer acknowledgements are merely an attempt to improve the quality of service, not make it perfect.

The *N-RESET* primitives are used to report catastrophes, such as crashes of either transport entity or the network service provider itself. After an *N-RESET* has been requested, indicated, responded to, and confirmed, the queues will be reset to their original empty state. Data present in the queues at the time of the *N-RESET* are lost. Again here, it is the job of the transport layer to recover from *N-RESET*s.

The OSI connectionless primitives are given in Fig. 5-4(b). The *N-UNITDATA* primitives are used to send up to 64,512 bytes of data (1K less than 2^{16}, to provide plenty of room for various headers and still keep the final unit less than 2^{16} bytes). The *N-UNITDATA* primitives provide no error control, no flow control, and no other control. The sender just dumps the packet into the subnet and hopes for the best.

format is used, as shown in Fig. 6-25. This TPDU is similar to those used in the connection-oriented service. The *Variable part* contains the source and destination TSAP addresses, and optionally a checksum.

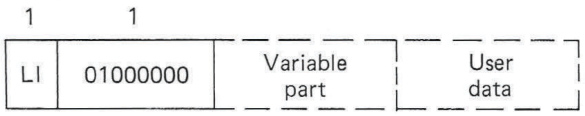


Fig. 6-25. The OSI connectionless TPDU format.

Work on the implementation of OSI and related transport protocols is discussed in (Chong, 1986; and Watson and Mamrak, 1987).

6.4.2. The Transport Layer in the ARPANET (TCP)

In the original ARPANET design, the subnet was assumed to offer virtual circuit service (i.e., be perfectly reliable). The first transport layer protocol, **NCP** (**Network Control Protocol**), was designed with a perfect subnet in mind. It just passed TPDUs (called messages) to the network layer and assumed that they would all be delivered in order to the destination. Experience showed that the ARPANET was indeed reliable enough for this protocol to be completely satisfactory for traffic within the ARPANET itself.

However, as time went on, and the ARPANET grew into the ARPA Internet, which included many LANs, a packet radio subnet, and several satellite channels, the end-to-end reliability of the subnet decreased. This development forced a major change in the transport layer and led to the gradual introduction of a new transport layer protocol, **TCP** (**Transmission Control Protocol**), which was specifically designed to tolerate an unreliable subnet (type C in OSI terms). Associated with TCP was a new network layer protocol, **IP**, which we studied in the previous chapter. Today TCP/IP is not only used in the ARPANET and ARPA Internet, but in many commercial systems as well.

A TCP transport entity accepts arbitrarily long messages from user processes, breaks them up into pieces not exceeding 64K bytes, and sends each piece as a separate datagram. The network layer gives no guarantee that datagrams will be delivered properly, so it is up to TCP to time out and retransmit them as need be. Datagrams that do arrive may well do so in the wrong order; it is also up to TCP to reassemble them into messages in the proper sequence.

Every byte of data transmitted by TCP has its own private sequence number. The sequence number space is 32 bits wide to make sure that old duplicates have long since vanished by the time the sequence numbers have wrapped around. TCP does, however, explicitly deal with the problem of delayed duplicates when attempting to establish a connection, using the three-way handshake for this purpose.

Figure 6-26 shows the header used by TCP. The first thing that strikes one is

that the minimum TCP header is 20 bytes. Unlike OSI Class 4, with which it is roughly comparable, TCP has only one TPDU header format. Let us dissect this large header field by field. The *Source port* and *Destination port* fields identify the end points of the connection (the TSAP addresses in OSI terminology). Each host may decide for itself how to allocate its ports.

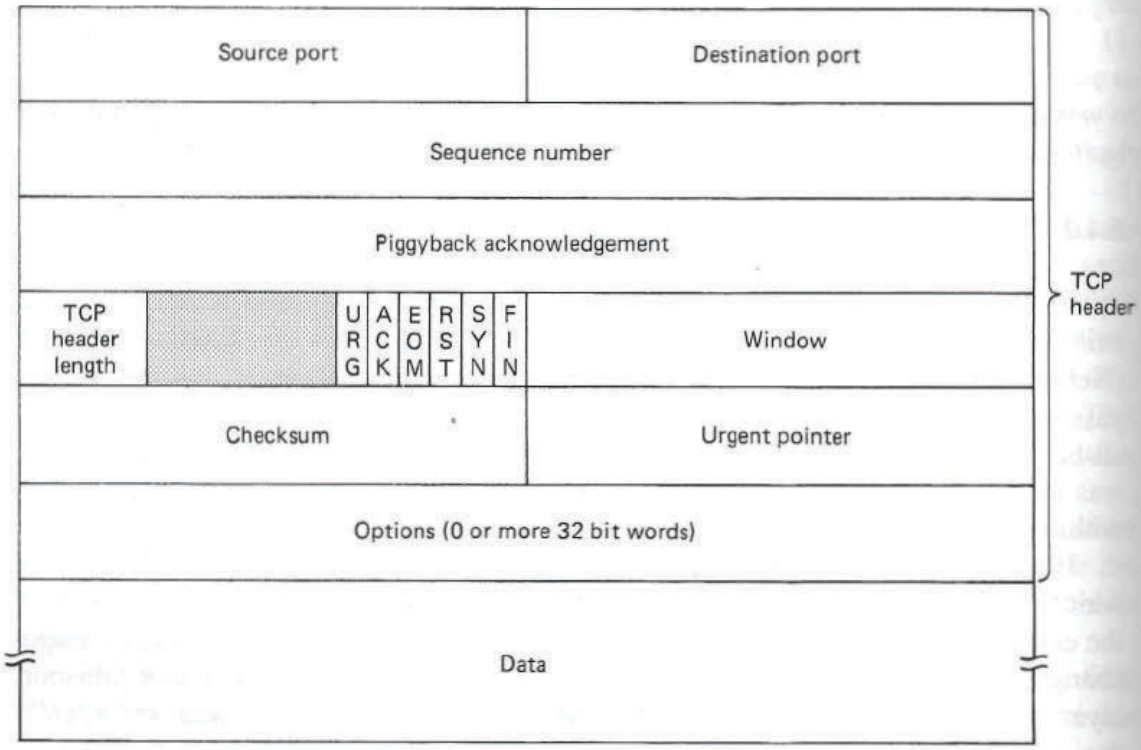


Fig. 6-26. The TCP TPDU structure.

The *Sequence number* and *Piggyback acknowledgement* fields perform their usual functions. They are 32 bits long because every byte of data is numbered in TCP.

The *TCP header length* tells how many 32-bit words are contained in the TCP header. This information is needed because the *Options* field is variable length, so the header is too.

Next come six 1-bit flags. *URG* is set to 1 if the *Urgent pointer* is in use. The *Urgent pointer* is used to indicate a byte offset from the current sequence number at which urgent data are to be found. This facility is in lieu of interrupt messages. The *SYN* bit is used to establish connections. The connection request has $SYN = 1$ and $ACK = 0$ to indicate that the piggyback acknowledgement field is not in use. The connection reply does bear an acknowledgement, so it has $SYN = 1$ and $ACK = 1$. In essence the *SYN* bit is used to denote *CONNECTION REQUEST* and *CONNECTION CONFIRM*, with the *ACK* bit used to distinguish between those two possibilities. The *FIN* bit is used to release a connection. It specifies that the

sender has no more data. After closing a connection, a process may continue to receive data indefinitely. The *RST* bit is used to reset a connection that has become confused due to delayed duplicate *SYN*s or host crashes. The *EOM* bit indicates End Of Message.

Flow control in TCP is handled using a variable-size sliding window. A 16-bit field is needed, because *Window* tells how many bytes may be sent beyond the byte acknowledged, not how many TPDUs.

A *Checksum* is also provided for extreme reliability. The checksum algorithm is simply to add up all the data, regarded as 16-bit words, and then to take the 1's complement of the sum.

The *Options* field is used for miscellaneous things, for example to communicate buffer sizes during the setup procedure.

Unlike most of the non-OSI protocols, TCP has a well-defined service interface. Calls are provided to actively and passively initiate connections (effectively, the *CONNECT* and *LISTEN* primitives from our example transport protocol, except that the *LISTEN* has two variants, one to accept connection requests from any caller and one to accept connection requests only from some specified caller). Calls are also provided to send and receive data, gracefully and abruptly terminate connections, and ask for the status of a connection.

Like OSI, the ARPANET also provides a connectionless transport protocol, **UDP (User Datagram Protocol)**. Like its OSI counterpart, it allows users to send messages without connection establishment and without any guarantee of delivery or sequencing. In effect, UDP is simply a user interface to IP. Its format is shown in Fig. 6-27.

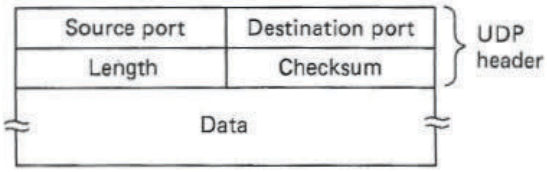


Fig. 6-27. The ARPANET UDP format.

Comparison of OSI Class 4 and TCP

The OSI Class 4 transport protocol (often called **TP4**), and TCP have numerous similarities but also some differences. In this section we will examine these similarities and differences (Groenbaek, 1986). Let us start with the points on which the two protocols are the same. Both protocols are designed for providing a reliable, connection-oriented, end-to-end transport service on top of an unreliable network that can lose, garble, store, and duplicate packets. Both must deal with the

SECOND EDITION

COMPUTER NETWORKS

By Andrew S. Tanenbaum

A comprehensive introduction to computer networks, this book emphasizes network protocols and algorithms, from the physical layer to the application layer and from local area networks to satellite networks. Assuming only a general familiarity with computer systems and programming, **Computer Networks** presents the full spectrum of basic protocols, concepts, algorithms, software, and technologies.

Revised to reflect the latest advances in the field, this second edition features material on LANs (incuding IEEE 802), ISDN (Integrated Services Digital Networks), and fiber optics networks. It also has detailed coverage of the upper layers of the OSI model (transport, session, presentation, and application) as well as extended discussions of MAP, TOP, and USENET.

Outstanding content highlights include:

• The Physical Layer • The Data Link Layer II—Medium Access • The Data Link Layer II—Protocols • The Network Layer • The Transport Layer • The Session Layer • The Presentation Layer • The Application Layer • Suggested Reading and Bibliography •

Other fine titles by Andrew S. Tanenbaum available from Prentice Hall:

Structured Computer Organization, Second Edition

This revised edition of a best-selling text extends the concept of structured programming to computer organization. By presenting the computer as a hierarchy of distinct levels, it illustrates how the diverse aspects of the computer relate to one another, and how topics such as chips, the microcomputer, micro-programming, machine language programming, operating systems, virtual machines, and multilevel computers can be integrated into a simple conceptual framework.

Operating Systems: Design and Implementation

This complete step-by-step guide to the principles and practices of today's operating systems examines interprocess communication, scheduling and paging algorithms, deadlocks, input/output, memory management, file systems, and security. To illustrate these principles, this practical text also contains the MINIX operating system source code (developed by the author), enabling readers to apply the general principles of operating systems to a UNIX-like system.

PRENTICE HALL, Englewood Cliffs, N.J. 07632

ISBN 0-13-162959-X