Internetworking With TCP/IP

Vol I:

Principles, Protocols, and Architecture

Third Edition

DOUGLAS E. COMER

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

PRENTICE HALL
Upper Saddle River, New Jersey 07458

Library of Congress Cataloging-in-Publication Data

Comer, Douglas
Internetworking with TCP/IP / Douglas E. Comer. -- 3rd ed.
p. cm.
Includes bibliographical references and index.
Contents: v. 1. Principles, protocols, and architecture
ISBN 0-13-216987-8 (v. 1)
1. TCP/IP (Computer network protocol) 2. Client/server computing.
3. Internetworking (Telecommunication) I. Title.
TK5105.585.C66 1995
005.2--dc20 95-1830
CIP

Acquisitions editor: ALAN APT
Production editor: IRWIN ZUCKER
Cover designer: WENDY ALLING JUDY
Buyer: LORI BULWIN
Editorial assistant: SHIRLEY MCGUIRE

Printed in the United States of America

20 19 18 17 16 15 14 13 12 11

ISBN 0-13-216987-8

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*
Pearson Education Asia Pte. Ltd., *Singapore*
Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*

To Chris

Additional Enthusiastic Comments About
Internetworking With TCP/IP Volume 1

"Unquestionably THE reference for TCP/IP; both informative and easy to read, this book is liked by both novice and experienced."

– Raj Yavatkar
University of Kentucky
US Editor, Computer Communications

"The third edition maintains Comer's Internetworking with TCP/IP as the acknowledged leader in TCP/IP books by adding up-to-the-minute material on ATM, CIDR, firewalls, DHCP and the next version of IP, IPng."

– Ralph Droms
Bucknell University
IFTF Working Group Chair

"Doug Comer remains the first and best voice of Internet technology. Despite the legion of 'Internet carpetbaggers' (the current crop of 'authors' who can barely spell F-T-P) which contributes noise – but no knowledge – on the Internet and its infrastructure, Dr. Comer shines through as the premiere source for lucid explanations and accurate information. He sets a standard for which many strive, but precious few attain."

– Marshall Rose
Dover Beach Consulting
IETF Area Director

"Comer's Volume 1 drastically changed the course of networking history."

– Dan Lynch
Interop Company
IAB Member

"When you need to teach the details of TCP/IP, you need the latest information. Once again, Comer separates the chaff from the wheat with his latest edition of the TCP/IP book that a generation of networkers grew up with."

– Shawn Ostermann
Ohio University

Contents

2

Review Of Underlying Network Technologies

2.1 Introduction

It is important to understand that the Internet is not a new kind of physical network. It is, instead, a method of interconnecting physical networks and a set of conventions for using networks that allow the computers they reach to interact. While network hardware plays only a minor role in the overall design, understanding the internet technology requires one to distinguish between the low-level mechanisms provided by the hardware itself and the higher-level facilities that the TCP/IP protocol software provides. It is also important to understand how the facilities supplied by packet-switched technology affect our choice of high-level abstractions.

This chapter introduces basic packet-switching concepts and terminology, and then reviews some of the underlying network hardware technologies that have been used in TCP/IP internets. Later chapters describe how these networks are interconnected and how the TCP/IP protocols accommodate vast differences in the hardware. While the list presented here is certainly not comprehensive, it clearly demonstrates the variety among physical networks over which TCP/IP operates. The reader can safely skip many of the technical details, but should try to grasp the idea of packet switching and try to imagine building a homogeneous communication system using such heterogeneous hardware. Most important, the reader should look closely at the details of the physical address schemes the various technologies use; later chapters will discuss in detail how high-level protocols use physical addresses.

2.2 Two Approaches To Network Communication

Whether they provide connections between one computer and another or between terminals and computers, communication networks can be divided into two basic types: *circuit-switched* (sometimes called *connection oriented*) and *packet-switched*† (sometimes called *connectionless*). Circuit-switched networks operate by forming a dedicated connection (circuit) between two points. The U.S. telephone system uses circuit switching technology – a telephone call establishes a circuit from the originating phone through the local switching office, across trunk lines, to a remote switching office, and finally to the destination telephone. While a circuit is in place, the phone equipment samples the microphone repeatedly, encodes the samples digitally, and transmits them across the circuit to the receiver. The sender is guaranteed that the samples can be delivered and reproduced because the circuit provides a guaranteed data path of 64 Kbps (thousand bits per second), the rate needed to send digitized voice. The advantage of circuit switching lies in its guaranteed capacity: once a circuit is established, no other network activity will decrease the capacity of the circuit. One disadvantage of circuit switching is cost: circuit costs are fixed, independent of traffic. For example, one pays a fixed rate for a phone call, even when the two parties do not talk.

Packet-switched networks, the type usually used to connect computers, take an entirely different approach. In a packet-switched network, data to be transferred across a network is divided into small pieces called *packets* that are multiplexed onto high capacity intermachine connections. A packet, which usually contains only a few hundred bytes of data, carries identification that enables the network hardware to know how to send it to the specified destination. For example, a large file to be transmitted between two machines must be broken into many packets that are sent across the network one at a time. The network hardware delivers the packets to the specified destination, where software reassembles them into a single file again. The chief advantage of packet-switching is that multiple communications among computers can proceed concurrently, with intermachine connections shared by all pairs of machines that are communicating. The disadvantage, of course, is that as activity increases, a given pair of communicating computers receives less of the network capacity. That is, whenever a packet switched network becomes overloaded, computers using the network must wait before they can send additional packets.

Despite the potential drawback of not being able to guarantee network capacity, packet-switched networks have become extremely popular. The motivations for adopting packet switching are cost and performance. Because multiple machines can share the network hardware, fewer connections are required and cost is kept low. Because engineers have been able to build high speed network hardware, capacity is not usually a problem. So many computer interconnections use packet-switching that, throughout the remainder of this text, the term *network* will refer only to packet-switched networks.

†In fact, it is possible to build hybrid hardware technologies; for our purposes, only the difference in functionality is important.

2.3 Wide Area And Local Area Networks

Packet-switched networks that span large geographical distances (e.g., the continental U.S.) are fundamentally different from those that span short distances (e.g., a single room). To help characterize the differences in capacity and intended use, packet switched technologies are often divided into two broad categories: *wide area networks* (*WAN*s) and *Local Area Networks* (*LAN*s). The two categories do not have formal definitions. Instead, vendors apply the terms loosely to help customers distinguish among technologies.

WAN technologies, sometimes called *long haul networks*, provide communication over large distances. Most WAN technologies do not limit the distance spanned; a WAN can allow the endpoints of a communication to be arbitrarily far apart. For example, a WAN can span a continent or can join computers across an ocean. Usually, WANs operate at slower speeds than LANs, and have much greater delay between connections. Typical speeds for a WAN range from 56 Kbps to 155 Mbps (million bits per second). Delays across a WAN can vary from a few milliseconds to several tenths of a second†

LAN technologies provide the highest speed connections among computers, but sacrifice the ability to span large distances. For example, a typical LAN spans a small area like a single building or a small campus and operates between 10 Mbps and 2 Gbps (billion bits per second). Because LAN technologies cover short distances, they offer lower delays than WANs. The delay across a LAN can be as short as a few tenths of a millisecond, or as long as 10 milliseconds.

We have already mentioned the general tradeoff between speed and distance: technologies that provide higher speed communication operate over shorter distances. There are other differences among technologies in the categories as well. In LAN technologies, each computer usually contains a network interface device that connects the machine directly to the network medium (e.g., a copper wire or coaxial cable). Often, the network itself is passive, depending on electronic devices in the attached computers to generate and receive the necessary electrical signals. In WAN technologies, a network usually consists of a series of complex computers called *packet switches* interconnected by communication lines and modems. The size of the network can be extended by adding a new switch and another communication line. Attaching a user's computer to a WAN means connecting it to one of the packet switches. Each switch along a path in the WAN introduces a delay when it receives a packet and forwards it to the next switch. Thus, the larger the WAN becomes the longer it takes to route traffic across it.

This book discusses software that hides the technological differences between networks and makes interconnection independent of the underlying hardware. To appreciate design choices in the software, it is necessary to understand how it relates to network hardware. The next sections present examples of network technologies that have been used in the Internet, showing some of the differences among them. Later chapters show how the TCP/IP software isolates such differences and makes the communication system independent of the underlying hardware technology.

†Such long delays result from WANs that communicate by sending signals to a satellite orbiting the earth.

2.3.1 Network Hardware Addresses

Each network hardware technology defines an *addressing mechanism* that computers use to specify the destination for each packet. Every computer attached to a network is assigned a unique address, usually an integer. A packet sent across a network includes a *destination address field* that contains the address of the intended recipient. The destination address appears in the same location in all packets, making it possible for the network hardware to examine the destination address easily. A sender must know the address of the intended recipient, and must place the recipient's address in the destination address field of a packet before transmitting the packet.

Each hardware technology specifies how computers are assigned addresses. The hardware specifies, for example, the number of bits in the address as well as the location of the destination address field in a packet. Although some technologies use compatible addressing schemes, many do not. This chapter contains a few examples of hardware addressing schemes; later chapters explain how TCP/IP accommodates diverse hardware addressing schemes.

2.4 Ethernet Technology

Ethernet is the name given to a popular packet-switched LAN technology invented at Xerox PARC in the early 1970s. Xerox Corporation, Intel Corporation, and Digital Equipment Corporation standardized Ethernet in 1978; IEEE released a compatible version of the standard using the number 802.3. Ethernet has become a popular LAN technology; most medium or large corporations use Ethernets. Because Ethernet is so popular, many variants exist; we will discuss the original design first and then cover variants.

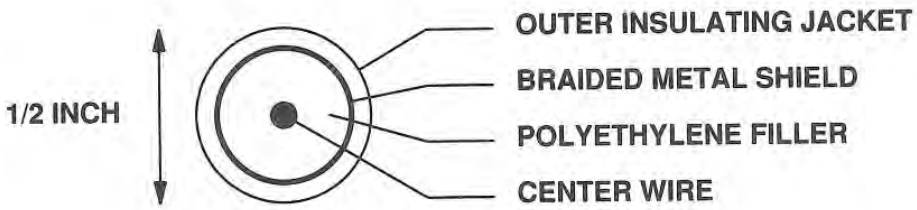

Figure 2.1 A cross-section of the coaxial cable used in the original Ethernet.

Each Ethernet cable is about 1/2 inch in diameter and up to 500 meters long. A resistor is added between the center wire and shield at each end to prevent reflection of electrical signals.

The original Ethernet design used a coaxial cable as Figure 2.1 illustrates. Called the *ether*, the cable itself is completely passive; all the active electronic components that make the network function are associated with computers that are attached to the network.

The connection between a computer and a coaxial Ethernet cable requires a hardware device called a *transceiver*. Physically, the connection between a transceiver and the Ethernet requires a small hole in the outer layers of the cable as Figure 2.2 illustrates. Technicians often use the term *tap* to describe the connection between an Ethernet transceiver and the cable. Usually, small metal pins mounted in the transceiver go through the hole and provide electrical contacts to the center wire and the braided shield. Some manufacturers' connectors require that the cable be cut and a "T" inserted.

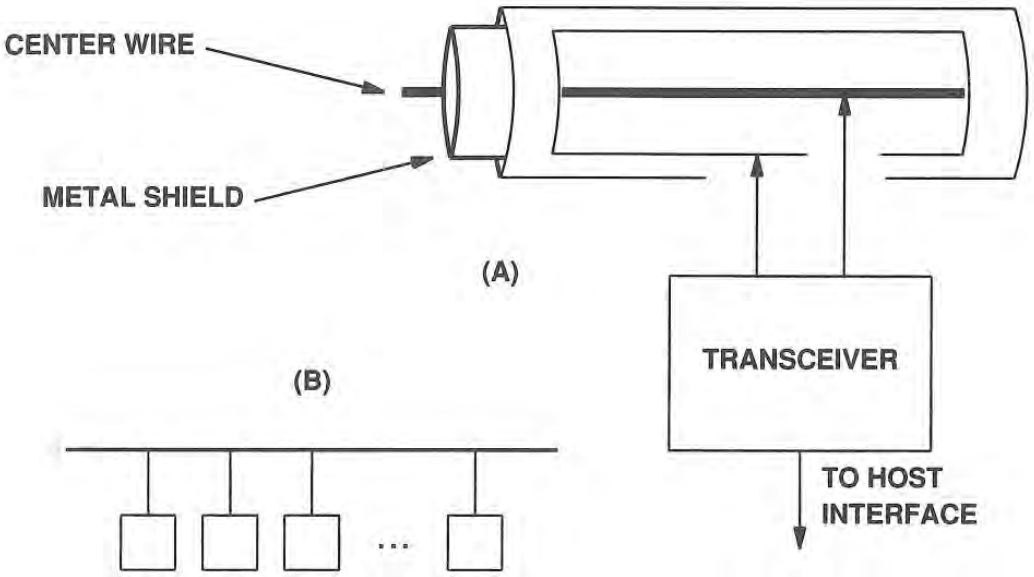

Figure 2.2 (a) A cutaway view of an Ethernet cable showing the details of electrical connections between a transceiver and the cable, and (b) the schematic diagram of an Ethernet with many computers connected.

Each connection to an Ethernet has two major electronic components. A *transceiver* connects to the center wire and braided shield on the cable, sensing and sending signals on the ether. A *host interface* or *host adapter* plugs into the computer's bus (e.g., on a motherboard) and connects to the transceiver.

A transceiver is a small piece of hardware usually found physically adjacent to the ether. In addition to the analog hardware that senses and controls electrical signals on the ether, a transceiver contains digital circuitry that allows it to communicate with a digital computer. The transceiver can sense when the ether is in use and can translate analog electrical signals on the ether to (and from) digital form. A cable called the *Attachment Unit Interface* (AUI) cable connects the transceiver to an adapter board in a host computer. Informally called a *transceiver cable*, the AUI cable contains many wires. The wires carry the electrical power needed to operate the transceiver, the sig-

nals that control the transceiver operation, and the contents of the packets being sent or received. Figure 2.3 illustrates how the components form a connection between a bus in a computer system and an Ethernet cable.

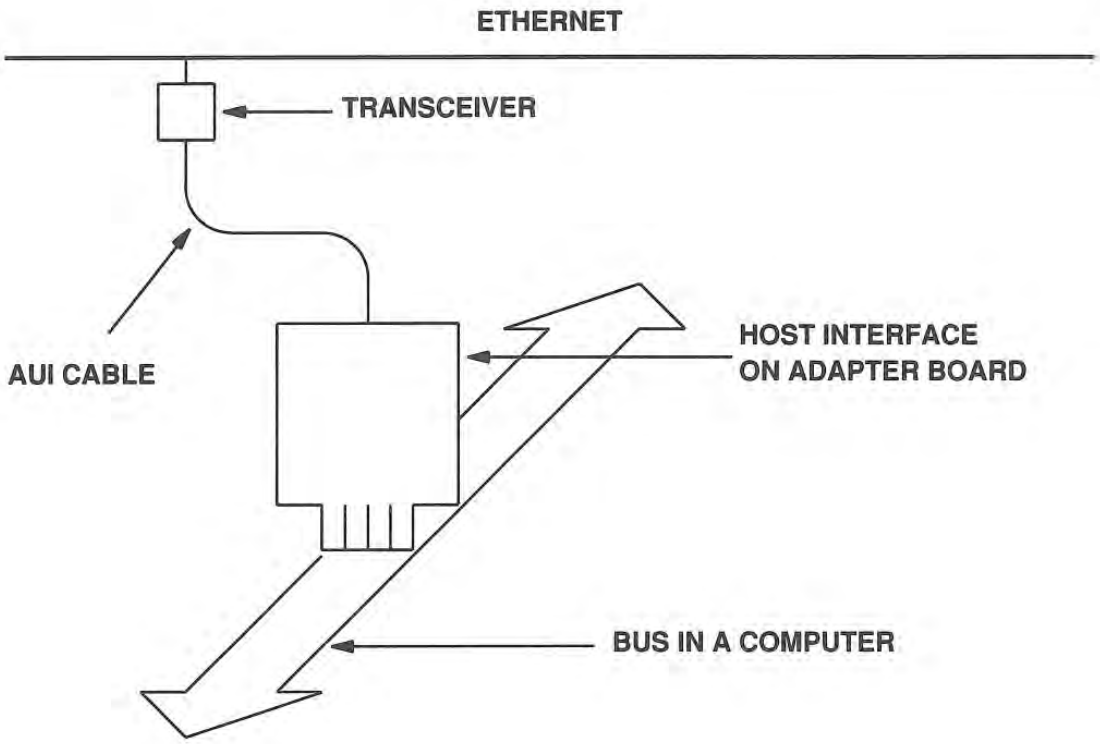

Figure 2.3 The two main electronic components that form a connection between a computer's bus and an Ethernet. The AUI cable that connects the host interface to the transceiver carries power and signals to control transceiver operation as well as packets being transmitted or received.

Each host interface controls the operation of one transceiver according to instructions it receives from the computer software. To the operating system software, the interface appears to be an input/output device that accepts basic data transfer instructions from the computer, controls the transceiver to carry them out, interrupts when the task has been completed, and reports status information. Although the transceiver is a simple hardware device, the host interface can be complex (e.g., it may contain a microprocessor used to control transfers between the computer memory and the ether).

In practice, organizations that use the original Ethernet in a conventional office environment run the Ethernet cable along the ceiling in each hall, and arrange for a connection from each office to attach to the cable. Figure 2.4 illustrates the resulting physical wiring scheme.

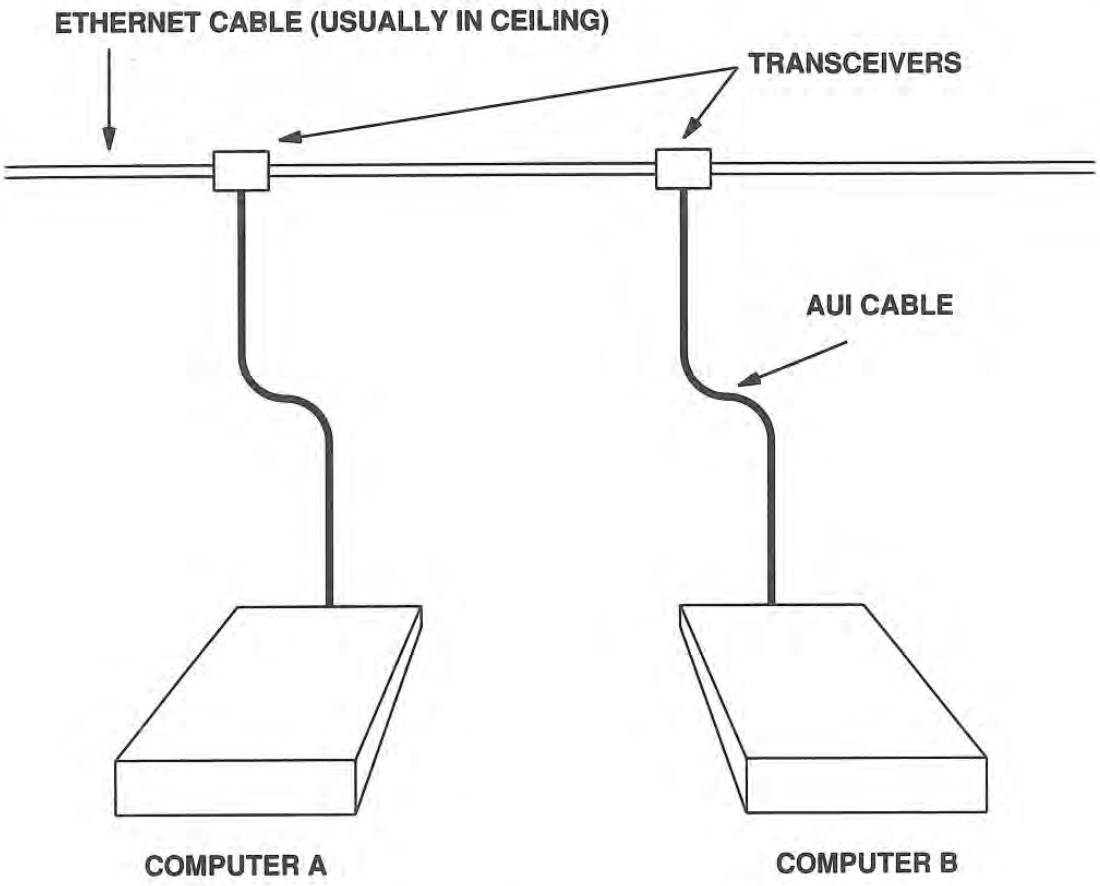

Figure 2.4 The physical connection of two computers to an Ethernet using the original wiring scheme. In an office environment, the Ethernet cable is usually placed in the hallway ceiling; each office has an AUI cable that connects a computer in the office to a transceiver attached to the Ethernet cable.

2.4.1 Thin-Wire Ethernet

Several components of the original Ethernet technology have undesirable properties. For example because a transceiver contains electronic components, it has a non-trivial cost. Furthermore, because transceivers are located with the cable and not with computers, they can be difficult to access or replace. The coaxial cable that forms the ether can also be difficult to install. In particular, to provide maximum protection against electrical interference from devices like electric motors, the cable contains heavy shielding that makes it difficult to bend. Finally, an AUI cable is also thick and difficult to bend.

To reduce costs for environments like offices that do not contain much electrical interference, engineers developed an alternative Ethernet wiring scheme. Called *thin-wire Ethernet* or *thinnet*†, the alternative coaxial cable is thinner, less expensive, and more flexible. However, a thin-wire Ethernet has some disadvantages. Because it does not provide as much protection from electrical interference, thin-wire Ethernet cannot be placed adjacent to powerful electrical equipment like that found in a factory. Furthermore, thin-wire Ethernet covers somewhat shorter distances and supports fewer computer connections per network than thick Ethernet.

To further reduce costs with thin-wire Ethernet, engineers replaced the costly transceiver with special high-speed digital circuits, and provided a direct connection from a computer to the ether. Thus, in a thin-wire scheme, a computer contains both the host interface and the circuitry that connects to the cable. Manufacturers of small computers and workstations find thin-wire Ethernet an especially attractive scheme because they can integrate Ethernet hardware into single board computers and mount connectors directly on the back of the computer.

Because a thin-wire Ethernet connects directly from one computer to another, the wiring scheme works well when many computers occupy a single room. The thin-wire cable runs directly from one computer to the next. To add a new computer, one only needs to link it into the chain. Figure 2.5 illustrates the connections used with thin-wire Ethernet.

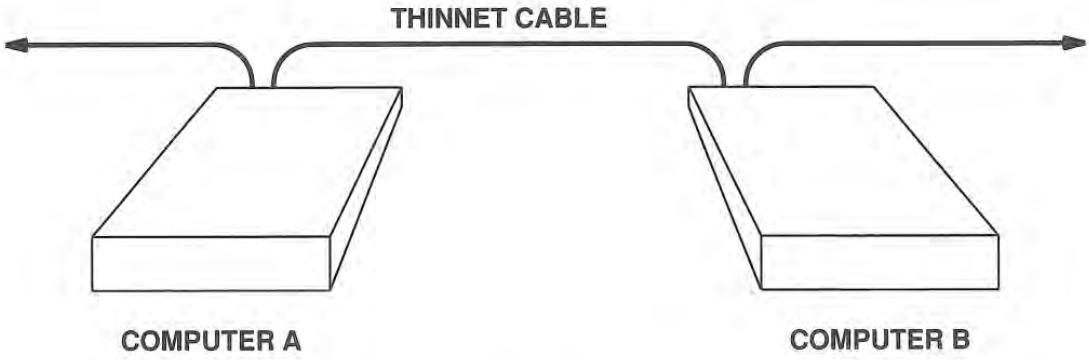

Figure 2.5 The physical connection of two computers using the thinnet wiring scheme. The ether passes directly from one computer to another; no external transceiver hardware is required.

Thin-wire Ethernets are designed to be easy to connect and disconnect. Thin-wire uses *BNC connectors*, which do not require tools to attach a computer to the cable. Thus, a user can connect a computer to a thin-wire Ethernet without the aid of a technician. Of course, allowing users to manipulate the ether has disadvantages: if a user disconnects the ether, it prevents all machines on the ether from communicating. In many situations, however, the advantages outweigh the disadvantages.

†To contrast it with thin-wire, the original Ethernet cable is sometimes called *thick Ethernet*, or *thicknet*.

2.4.2 Twisted Pair Ethernet

Advances in technology have made it possible to build Ethernets that do not need the electrical shielding of a coaxial cable. Called *twisted pair Ethernet*, the technology allows a computer to access an Ethernet using a pair of conventional unshielded copper wires similar to the wires used to connect telephones. The advantages of using twisted pair wiring are that it further reduces costs and protects other computers on the network from a user who disconnects a single computer. In some cases, a twisted pair technology can make it possible for an organization to use Ethernet over existing telephone wiring without adding new cables.

Known by the technical name *10Base-T*, the twisted pair wiring scheme connects each computer to an Ethernet *hub* as Figure 2.6 shows.

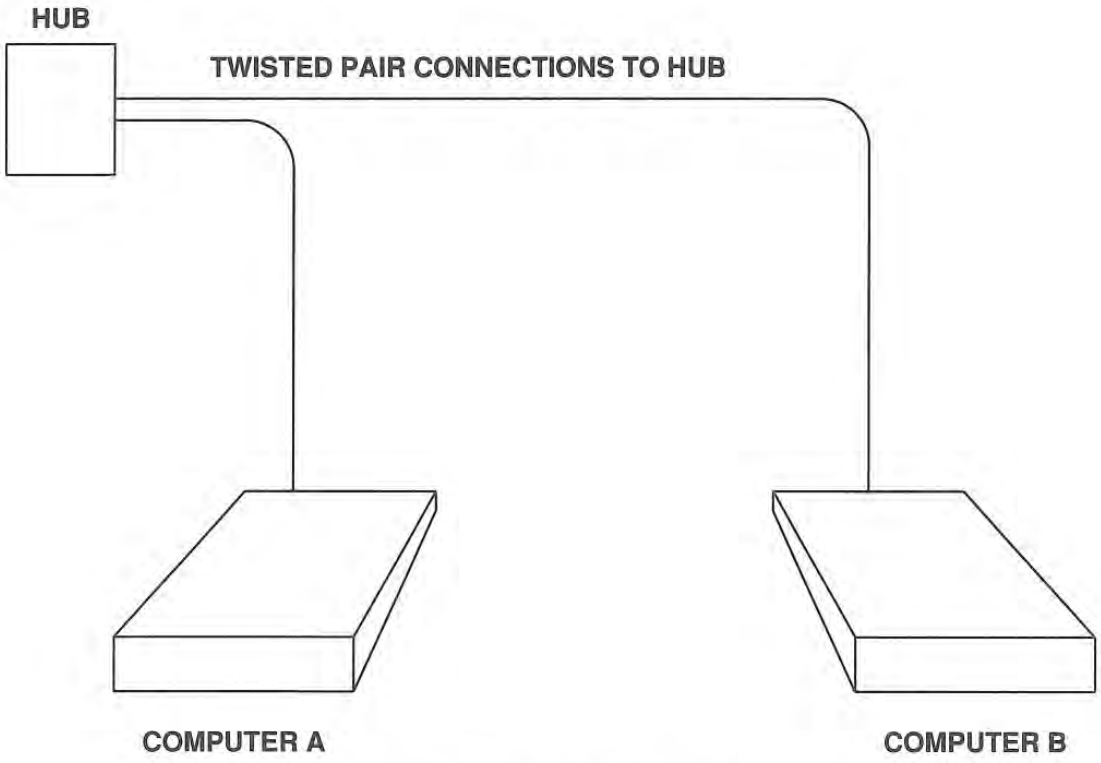

Figure 2.6 An illustration of Ethernet using twisted pair wiring. Each computer connects to a hub over a conventional pair of wires.

The hub is an electronic device that simulates the signals on an Ethernet cable. Physically, a hub consists of a small box that usually resides in a wiring closet; a connection between a hub and a computer must be less than 100 meters long. A hub requires power, and can allow authorized personnel to monitor and control its operation over the network. To the host interface in a computer, a connection to a hub appears to

operate the same way as a connection to a transceiver. That is, an Ethernet hub provides the same communication capability as a thick or thin Ethernet; hubs merely offer an alternative wiring scheme.

2.4.3 Adapters And Multiple Wiring Schemes

A connection to thick Ethernet requires an AUI connector, a connection to thin-wire Ethernet requires a BNC connector, and a connection to 10Base-T requires an RJ45 connector that resembles the modular connectors used with telephones. Many Ethernet products allow each customer to choose a wiring scheme. For example, adapter boards for personal computers often come with three connectors as Figure 2.7 illustrates. Although only one connector can be used at any time, a computer that has such an adapter can be moved from one wiring scheme to another easily.

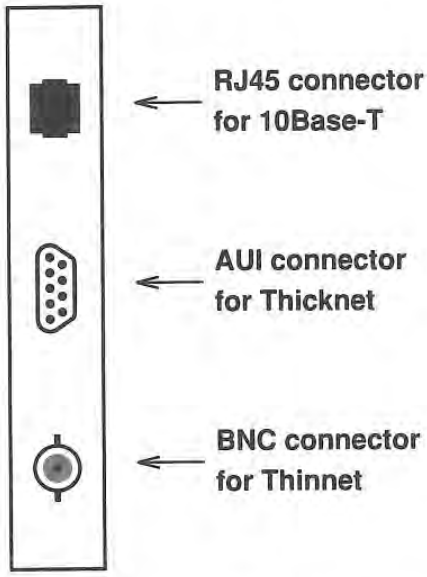

Figure 2.7 A typical Ethernet adapter card with three connectors for the three Ethernet wiring schemes. Although the adapter contains three connectors, it can only use one wiring scheme at any time.

2.4.4 Properties of an Ethernet

The Ethernet is a 10 Mbps broadcast bus technology with best-effort delivery semantics and distributed access control. It is a *bus* because all stations share a single communication channel; it is *broadcast* because all transceivers receive every transmission. The method used to direct packets from one station to just one other station or a subset of all stations will be discussed later. For now, it is enough to understand that

transceivers do not distinguish among transmissions – a transceiver passes all packets from the cable to the host interface, which chooses packets the computer should receive and filters out all others. Ethernet is called a *best-effort delivery* mechanism because the hardware provides no information to the sender about whether the packet was delivered. For example, if the destination machine happens to be powered down, packets sent to it will be lost, and the sender will not be notified. We will see later how the TCP/IP protocols accommodate best-effort delivery hardware.

Ethernet access control is distributed because, unlike some network technologies, Ethernet has no central authority to grant access. The Ethernet access scheme is called *Carrier Sense Multiple Access* with *Collision Detect* (*CSMA/CD*). It is *CSMA* because multiple machines can access the Ethernet simultaneously and each machine determines whether the ether is idle by sensing whether a carrier wave is present. When a host interface has a packet to transmit, it listens to the ether to see if a message is being transmitted (i.e., performs carrier sensing). When no transmission is sensed, the host interface starts transmitting. Each transmission is limited in duration (because there is a maximum packet size). Furthermore, the hardware must observe a minimum idle time between transmissions, which means that no single pair of communicating machines can use the network without giving other machines an opportunity for access.

2.4.5 Collision Detection And Recovery

When a transceiver begins transmission, the signal does not reach all parts of the network simultaneously. Instead it travels along the cable at approximately 80% of the speed of light. Thus, it is possible for two transceivers to both sense that the network is idle and begin transmission simultaneously. When the two electrical signals cross they become scrambled, such that neither is meaningful. Such incidents are called *collisions*.

The Ethernet handles collisions in an ingenious fashion. Each transceiver monitors the cable while it is transmitting to see if a foreign signal interferes with its transmission. Technically, the monitoring is called *collision detect* (*CD*), making the Ethernet a CSMA/CD network. When a collision is detected, the host interface aborts transmission, waits for activity to subside, and tries again. Care must be taken or the network could wind up with all transceivers busily attempting to transmit and every transmission producing a collision. To help avoid such situations, Ethernet uses a binary exponential backoff policy where a sender delays a random time after the first collision, twice as long if a second attempt to transmit also produces a collision, four times as long if a third attempt results in a collision, and so on. The motivation for exponential backoff is that in the unlikely event many stations attempt to transmit simultaneously, a severe traffic jam could occur. In such a jam, there is a high probability two stations will choose random backoffs that are close together. Thus, the probability of another collision is high. By doubling the random delay, the exponential backoff strategy quickly spreads the stations' attempts to retransmit over a reasonably long period of time, making the probability of further collisions extremely small.

2.4.6 Ethernet Capacity

The standard Ethernet is rated at 10 Mbps, which means that data can be transmitted onto the cable at 10 million bits per second. Although a computer can generate data at Ethernet speed, raw network speed should not be thought of as the rate at which two computers can exchange data. Instead, network speed should be thought of as a measure of network total traffic capacity. Think of a network as a highway connecting multiple cities, and think of packets as cars on the highway. High bandwidth makes it possible to carry heavy traffic loads, while low bandwidth means the highway cannot carry as much traffic. A 10 Mbps Ethernet, for example, can handle a few computers that generate heavy loads, or many computers that generate light loads.

2.4.7 Ethernet Hardware Addresses

Ethernet defines a 48-bit addressing scheme. Each computer attached to an Ethernet network is assigned a unique 48-bit number known as its *Ethernet address*. To assign an address, Ethernet hardware manufacturers purchase blocks of Ethernet addresses† and assign them in sequence as they manufacture Ethernet interface hardware. Thus, no two hardware interfaces have the same Ethernet address.

Usually, the Ethernet address is fixed in machine readable form on the host interface hardware. Because Ethernet addresses belong to hardware devices, they are sometimes called *hardware addresses* or *physical addresses*. Note the following important property of Ethernet physical addresses:

> *Physical addresses are associated with the Ethernet interface hardware; moving the hardware interface to a new machine or replacing a hardware interface that has failed changes the machine's physical address.*

Knowing that Ethernet physical addresses can change will make it clear why higher levels of the network software are designed to accommodate such changes.

The host interface hardware examines packets and determines the packets that should be sent to the host. Recall that each interface receives a copy of every packet – even those addressed to other machines. The host interface uses the destination address field in a packet as a filter. The interface ignores those packets that are addressed to other machines, and passes to the host only those packets addressed to it. The addressing mechanism and hardware filter are needed to prevent a computer from being overwhelmed with incoming data. Although the computer's central processor could perform the check, doing so in the host interface keeps traffic on the Ethernet from slowing down processing on all computers.

A 48-bit Ethernet address can do more than specify a single destination computer. An address can be one of three types:

†The Institute for Electrical and Electronic Engineers (IEEE) manages the Ethernet address space and assigns addresses as needed.

- The physical address of one network interface (a *unicast* address)
- The network *broadcast* address
- A *multicast* address

By convention, the broadcast address (all 1s) is reserved for sending to all stations simultaneously. Multicast addresses provide a limited form of broadcast in which a subset of the computers on a network agree to listen to a given multicast address. The set of participating computers is called a *multicast group*. To join a multicast group, a computer must instruct its host interface to accept the group's multicast address. The advantage of multicasting lies in the ability to limit broadcasts: every computer in a multicast group can be reached with a single packet transmission, but computers that choose not to participate in a particular multicast group do not receive packets sent to the group.

To accommodate broadcast and multicast addressing, Ethernet interface hardware must recognize more than its physical address. A host interface usually accepts at least two kinds of packets: those addressed to the interface's physical (i.e., unicast) address and those addressed to the network broadcast address. Some interfaces can be programmed to recognize multicast addresses or even alternate physical addresses. When the operating system starts, it initializes the Ethernet interface, giving it a set of addresses to recognize. The interface then examines the destination address field in each packet, passing on to the host only those transmissions designated for one of the specified addresses.

2.4.8 Ethernet Frame Format

The Ethernet should be thought of as a link-level connection among machines. Thus, it makes sense to view the data transmitted as a *frame*†. Ethernet frames are of variable length, with no frame smaller than 64 octets‡ or larger than 1518 octets (header, data, and CRC). As in all packet-switched networks, each Ethernet frame contains a field that contains the address of its destination. Figure 2.8 shows that the Ethernet frame format contains the physical source address as well as the destination address.

Preamble	Destination Address	Source Address	Frame Type	Frame Data	CRC
8 octets	6 octets	6 octets	2 octets	64–1500 octets	4 octets

Figure 2.8 The format of a frame (packet) as it travels across an Ethernet preceded by a preamble. Fields are not drawn to scale.

In addition to identifying the source and destination, each frame transmitted across the Ethernet contains a *preamble*, *type field*, *data field*, and *Cyclic Redundancy Check* (*CRC*). The preamble consists of 64 bits of alternating *0*s and *1*s to help receiving

†The term *frame* derives from communication over serial lines in which the sender "frames" the data by adding special characters before and after the transmitted data.

‡Technically, the term *byte* refers to a hardware-dependent character size; networking professionals use the term *octet*, because it refers to an 8-bit quantity on all computers.

nodes synchronize. The 32-bit CRC helps the interface detect transmission errors: the sender computes the CRC as a function of the data in the frame, and the receiver recomputes the CRC to verify that the packet has been received intact.

The frame type field contains a 16-bit integer that identifies the type of the data being carried in the frame. From the Internet point of view, the frame type field is essential because it means Ethernet frames are *self-identifying*. When a frame arrives at a given machine, the operating system uses the frame type to determine which protocol software module should process the frame. The chief advantages of self-identifying frames are that they allow multiple protocols to be used together on a single machine and they allow multiple protocols to be intermixed on the same physical network without interference. For example, one could have an application program using Internet protocols while another used a local experimental protocol. The operating system uses the type field of an arriving frame to decide how to process the contents. We will see that the TCP/IP protocols use self-identifying Ethernet frames to distinguish among several protocols.

2.4.9 Extending An Ethernet With Repeaters

Although an Ethernet cable has a maximum length, the network can be extended in two ways: using repeaters and bridges. A hardware device called a *repeater* can be used to relay electrical signals from one cable to another. However, at most two repeaters can be placed between any two machines, so the total length of a single Ethernet is still rather short (three segments of 500 meters each). Figure 2.9 shows a typical use of repeaters in an office building. A single cable runs vertically up the building, and a repeater attaches the backbone to an additional cable on each floor. Computers attach to the cables on each floor.

2.4.10 Extending An Ethernet With Bridges

Bridges are superior to repeaters because they do not replicate noise, errors, or malformed frames; a completely valid frame must be received before the bridge will accept and transmit it on the other segment. Furthermore, bridge interfaces follow the Ethernet CSMA/CD rules, so collisions and propagation delays on one wire remain isolated from those on the other. As a result, an (almost) arbitrary number of Ethernets can be connected together with bridges. The important point is:

> *Bridges hide the details of interconnection: a set of bridged segments acts like a single Ethernet.*

A computer uses exactly the same hardware to communicate with a computer across a bridge as it uses to communicate with a computer on the local segment.

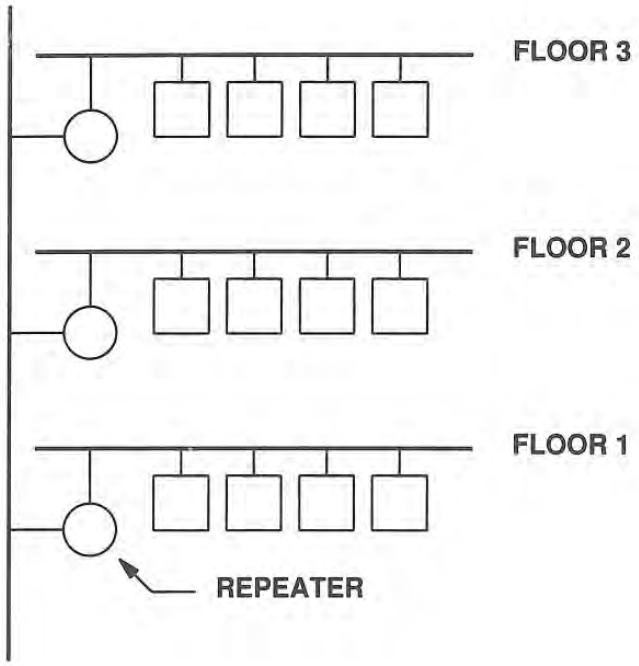

Figure 2.9 Repeaters used to join Ethernet cables in a building. At most two repeaters can be placed between a pair of communicating machines.

Most bridges do much more than replicate frames from one wire to another: they make intelligent decisions about which frames to forward. Such bridges are called *adaptive*, or *learning* bridges. An adaptive bridge consists of a computer with two Ethernet interfaces. The software in an adaptive bridge keeps two address lists, one for each interface. When a frame arrives from Ethernet E_1, the adaptive bridge adds the 48-bit Ethernet *source* address to the list associated with E_1. Similarly, when a frame arrives from Ethernet E_2, the bridge adds the source address to the list associated with E_2. Thus, over time the adaptive bridge will learn which machines lie on E_1 and which lie on E_2.

After recording the source address of a frame, the adaptive bridge uses the destination address to determine whether to forward the frame. If the address list shows that the destination lies on the Ethernet from which the frame arrived, the bridge does not forward the frame. If the destination is not in the address list (i.e., the destination is a broadcast or multicast address or the bridge has not yet learned the location of the destination), the bridge forwards the frame to the other Ethernet.

The advantages of adaptive bridges should be obvious. Because the bridge uses addresses found in normal traffic, it is completely automatic – humans need not configure the bridge with specific addresses. Because it does not forward traffic unnecessari-

ly, a bridge helps improve the performance of an overloaded network by isolating traffic on specific segments. Bridges work exceptionally well if a network can be divided physically into two segments that each contain a set of computers that communicate frequently (e.g., each segment contains a set of workstations along with a server, and the workstations direct most of their traffic to the server). To summarize:

> *An adaptive Ethernet bridge connects two Ethernet segments, forwarding frames from one to the other. It uses source addresses to learn which machines lie on which Ethernet segment, and it combines information learned with destination addresses to eliminate forwarding when unnecessary.*

From the TCP/IP point of view, bridged Ethernets are merely another form of physical network connection. The important point is:

> *Because the connection among physical cables provided by bridges and repeaters is transparent to machines using the Ethernet, we think of multiple Ethernet segments connected by bridges and repeaters as a single physical network system.*

Most commercial bridges are much more sophisticated and robust than our description indicates. When first powered up, they check for other bridges and learn the topology of the network. They use a distributed spanning-tree algorithm to decide how to forward frames. In particular, the bridges decide how to propagate broadcast packets so only one copy of a broadcast frame is delivered to each wire. Without such an algorithm, Ethernets and bridges connected in a cycle would produce catastrophic results because they would forward broadcast packets in both directions simultaneously.

2.5 Fiber Distributed Data Interconnect (FDDI)

FDDI is a popular local area networking technology that provides higher bandwidth than Ethernet. Unlike Ethernet and other LAN technologies that use cables to carry electrical signals, FDDI uses glass fibers and transfers data by encoding it in pulses of light†.

Optical fiber has two advantages over copper wire. First, because electrical noise does not interfere with an optical connection, the fiber can lie adjacent to powerful electrical devices. Second, because optical fibers use light, the amount of data that can be sent per unit time is much higher than cables that carry electrical signals.

It might seem that glass fibers would be difficult to install and would break if bent. However, an optical cable is surprisingly flexible. The glass fiber itself has an extremely small diameter, and the cable includes a plastic jacket that protects the fiber from breaking. Such a cable cannot bend at a ninety degree angle, but it can bend in an arc with a diameter of a few inches. Thus, installation is not difficult.

---

†A related technology known as *Copper Distributed Data Interface* (*CDDI*) works like FDDI, but uses copper cables to carry signals.

7

Internet Protocol: Connectionless Datagram Delivery

7.1 Introduction

Previous chapters review pieces of network hardware and software that make internet communication possible, explaining the underlying network technologies and address resolution. This chapter explains the fundamental principle of connectionless delivery and discusses how it is provided by the *Internet Protocol* (*IP*), one of the two major protocols used in internetworking. We will study the format of IP datagrams and see how they form the basis for all internet communication. The next two chapters continue our examination of the Internet Protocol by discussing datagram routing and error handling.

7.2 A Virtual Network

Chapter 3 discussed an internet architecture in which routers connect multiple physical networks. Looking at the architecture may be misleading, because the focus should be on the interface that an internet provides to users, not on the interconnection technology.

A user thinks of an internet as a single virtual network that interconnects all hosts, and through which communication is possible; its underlying architecture is both hidden and irrelevant.

In a sense, an internet is an abstraction of physical networks because, at the lowest level, it provides the same functionality: accepting packets and delivering them. Higher levels of internet software add most of the rich functionality users perceive.

7.3 Internet Architecture And Philosophy

Conceptually, a TCP/IP internet provides three sets of services as shown in Figure 7.1; their arrangement in the figure suggests dependencies among them. At the lowest level, a connectionless delivery service provides a foundation on which everything rests. At the next level, a reliable transport service provides a higher level platform on which applications depend. We will soon explore each of these services, understand what they provide, and see the protocols associated with them.

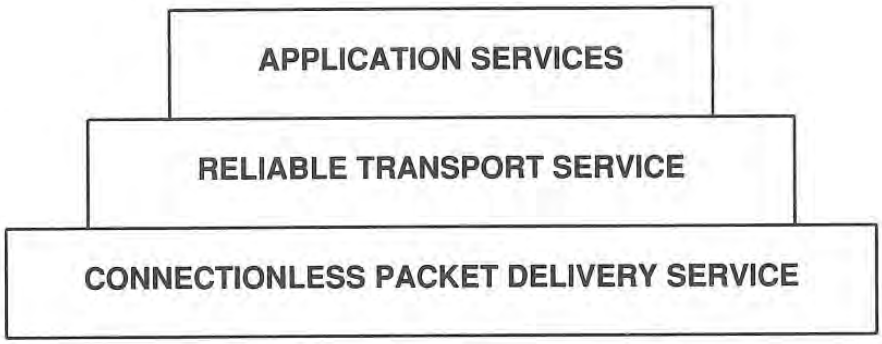

Figure 7.1 The three conceptual layers of internet services.

7.4 The Concept Of Unreliable Delivery

Although we can associate protocol software with each of the services in Figure 7.1, the reason for identifying them as conceptual parts of the internet is that they clearly point out the philosophical underpinnings of the design. The point is:

Internet software is designed around three conceptual networking services arranged in a hierarchy; much of its success has resulted because this architecture is surprisingly robust and adaptable.

One of the most significant advantages of this conceptual separation is that it becomes possible to replace one service without disturbing others. Thus, research and development can proceed concurrently on all three.

7.5 Connectionless Delivery System

The most fundamental internet service consists of a packet delivery system. Technically, the service is defined as an unreliable, best-effort, connectionless packet delivery system, analogous to the service provided by network hardware that operates on a best-effort delivery paradigm. The service is called *unreliable* because delivery is not guaranteed. The packet may be lost, duplicated, delayed, or delivered out of order, but the service will not detect such conditions, nor will it inform the sender or receiver. The service is called *connectionless* because each packet is treated independently from all others. A sequence of packets sent from one computer to another may travel over different paths, or some may be lost while others are delivered. Finally, the service is said to use *best-effort delivery* because the internet software makes an earnest attempt to deliver packets. That is, the internet does not discard packets capriciously; unreliability arises only when resources are exhausted or underlying networks fail.

7.6 Purpose Of The Internet Protocol

The protocol that defines the unreliable, connectionless delivery mechanism is called the *Internet Protocol* and is usually referred to by its initials, *IP*†. IP provides three important definitions. First, the IP protocol defines the basic unit of data transfer used throughout a TCP/IP internet. Thus, it specifies the exact format of all data as it passes across a TCP/IP internet. Second, IP software performs the *routing* function, choosing a path over which data will be sent. Third, in addition to the precise, formal specification of data formats and routing, IP includes a set of rules that embody the idea of unreliable packet delivery. The rules characterize how hosts and routers should process packets, how and when error messages should be generated, and the conditions under which packets can be discarded. IP is such a fundamental part of the design that a TCP/IP internet is sometimes called an *IP-based technology*.

We begin our consideration of IP in this chapter by looking at the packet format it specifies. We leave until later chapters the topics of routing and error handling.

7.7 The Internet Datagram

The analogy between a physical network and a TCP/IP internet is strong. On a physical network, the unit of transfer is a frame that contains a header and data, where the header gives information such as the (physical) source and destination addresses. The internet calls its basic transfer unit an *Internet datagram*, sometimes referred to as

†The abbreviation IP gives rise to the term "IP address."

an *IP datagram* or merely a *datagram*. Like a typical physical network frame, a datagram is divided into header and data areas. Also like a frame, the datagram header contains the source and destination addresses and a type field that identifies the contents of the datagram. The difference, of course, is that the datagram header contains IP addresses whereas the frame header contains physical addresses. Figure 7.2 shows the general form of a datagram:

Figure 7.2 General form of an IP datagram, the TCP/IP analogy to a network frame. IP specifies the header format including the source and destination IP addresses. IP does not specify the format of the data area; it can be used to transport arbitrary data.

7.7.1 Datagram Format

Now that we have described the general layout of an IP datagram, we can look at the contents in more detail. Figure 7.3 shows the arrangement of fields in a datagram:

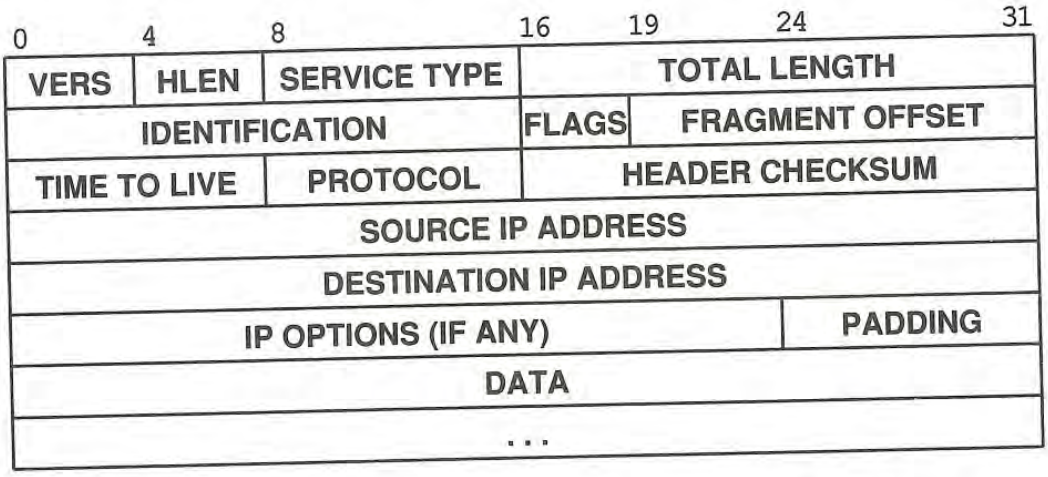

Figure 7.3 Format of an Internet datagram, the basic unit of transfer in a TCP/IP internet.

Because datagram processing occurs in software, the contents and format are not constrained by any hardware. For example, the first 4-bit field in a datagram (*VERS*) contains the version of the IP protocol that was used to create the datagram. It is used to verify that the sender, receiver, and any routers in between them agree on the format

of the datagram. All IP software is required to check the version field before processing a datagram to ensure it matches the format the software expects. If standards change, machines will reject datagrams with protocol versions that differ from theirs, preventing them from misinterpreting datagram contents according to an outdated format. The current IP protocol version is *4*.

The header length field (*HLEN*), also 4 bits, gives the datagram header length measured in 32-bit words. As we will see, all fields in the header have fixed length except for the *IP OPTIONS* and corresponding *PADDING* fields. The most common header, which contains no options and no padding, measures 20 octets and has a header length field equal to *5*.

The *TOTAL LENGTH* field gives the length of the IP datagram measured in octets, including octets in the header and data. The size of the data area can be computed by subtracting the length of the header (*HLEN*) from the *TOTAL LENGTH*. Because the *TOTAL LENGTH* field is 16 bits long, the maximum possible size of an IP datagram is 2^{16} or 65,535 octets. In most applications this is not a severe limitation. It may become more important in the future if higher speed networks can carry data packets larger than 65,535 octets.

7.7.2 Datagram Type Of Service And Datagram Precedence

Informally called *Type Of Service* (*TOS*), the 8-bit *SERVICE TYPE* field specifies how the datagram should be handled and is broken down into five subfields as shown in Figure 7.4:

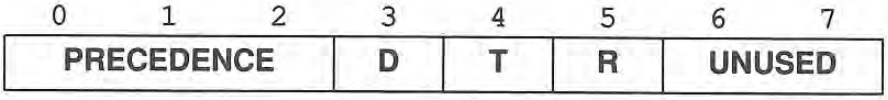

Figure 7.4 The five subfields that comprise the 8-bit *SERVICE TYPE* field.

Three *PRECEDENCE* bits specify datagram precedence, with values ranging from 0 (normal precedence) through 7 (network control), allowing senders to indicate the importance of each datagram. Although most host and router software ignores type of service, it is an important concept because it provides a mechanism that can allow control information to have precedence over data. For example, if all hosts and routers honor precedence, it is possible to implement congestion control algorithms that are not affected by the congestion they are trying to control.

Bits *D*, *T*, and *R* specify the type of transport the datagram desires. When set, the *D* bit requests low delay, the *T* bit requests high throughput, and the *R* bit requests high reliability. Of course, it may not be possible for an internet to guarantee the type of transport requested (i.e., it could be that no path to the destination has the requested property). Thus, we think of the transport request as a hint to the routing algorithms,

not as a demand. If a router does know more than one possible route to a given destination, it can use the type of transport field to select one with characteristics closest to those desired. For example, suppose a router can select between a low capacity leased line or a high bandwidth (but high delay) satellite connection. Datagrams carrying keystrokes from a user to a remote computer could have the *D* bit set requesting that they be delivered as quickly as possible, while datagrams carrying a bulk file transfer could have the *T* bit set requesting that they travel across the high capacity satellite path.

It is also important to realize that routing algorithms must choose from among underlying physical network technologies that each have characteristics of delay, throughput, and reliability. Often, a given technology trades off one characteristic for another (e.g., higher throughput rates at the expense of longer delay). Thus, the idea is to give the routing algorithm a hint about what is most important; it seldom makes sense to specify all three types of service. To summarize:

> *We regard the type of transport specification as a hint to the routing algorithm that helps it choose among various paths to a destination based on its knowledge of the hardware technologies available on those paths. An internet does not guarantee the type of transport requested.*

7.7.3 Datagram Encapsulation

Before we can understand the next fields in a datagram, it is important to consider how datagrams relate to physical network frames. We start with a question: "How large can a datagram be?" Unlike physical network frames that must be recognized by hardware, datagrams are handled by software. They can be of any length the protocol designers choose. We have seen that the current datagram format allots only 16 bits to the total length field, limiting the datagram to at most 65,535 octets. However, that limit could be changed in later versions of the protocol.

More fundamental limits on datagram size arise in practice. We know that as datagrams move from one machine to another, they must always be transported by the underlying physical network. To make internet transportation efficient, we would like to guarantee that each datagram travels in a distinct physical frame. That is, we want our abstraction of a physical network packet to map directly onto a real packet if possible.

The idea of carrying one datagram in one network frame is called *encapsulation*. To the underlying network, a datagram is like any other message sent from one machine to another. The hardware does not recognize the datagram format, nor does it understand the IP destination address. Thus, as Figure 7.5 shows, when one machine sends an IP datagram to another, the entire datagram travels in the data portion of the network frame.

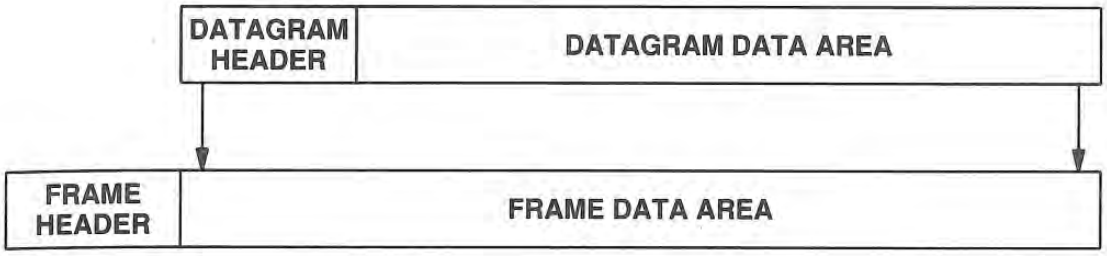

Figure 7.5 The encapsulation of an IP datagram in a frame. The physical network treats the entire datagram, including the header, as data.

7.7.4 Datagram Size, Network MTU, and Fragmentation

In the ideal case, the entire IP datagram fits into one physical frame, making transmission across the physical net efficient.† To achieve such efficiency, the designers of IP might have selected a maximum datagram size such that a datagram would always fit into one frame. But which frame size should be chosen? After all, a datagram may travel across many types of physical networks as it moves across an internet to its final destination.

To understand the problem, we need a fact about network hardware: each packet-switching technology places a fixed upper bound on the amount of data that can be transferred in one physical frame. For example, the Ethernet limits transfers to 1500‡ octets of data, while FDDI permits approximately 4470 octets of data per frame. We refer to these limits as the network's *maximum transfer unit* or *MTU*. MTU sizes can be quite small: some hardware technologies limit transfers to 128 octets or less. Limiting datagrams to fit the smallest possible MTU in the internet makes transfers inefficient when those datagrams pass across a network that can carry larger size frames. However, allowing datagrams to be larger than the minimum network MTU in an internet means that a datagram may not always fit into a single network frame.

The choice should be obvious: the point of the internet design is to hide underlying network technologies and make communication convenient for the user. Thus, instead of designing datagrams that adhere to the constraints of physical networks, TCP/IP software chooses a convenient initial datagram size and arranges a way to divide large datagrams into smaller pieces when the datagram needs to traverse a network that has a small MTU. The small pieces into which a datagram is divided are called *fragments*, and the process of dividing a datagram is known as *fragmentation*.

As Figure 7.6 illustrates, fragmentation usually occurs at a router somewhere along the path between the datagram source and its ultimate destination. The router receives a datagram from a network with a large MTU and must send it over a network for which the MTU is smaller than the datagram size.

---

†A field in the frame header usually identifies the data being carried; Ethernet uses the type value 0800_{16} to specify that the data area contains an encapsulated IP datagram.

‡The limit of 1500 comes from the Ethernet specification; when used with a SNAP header the IEEE 802.3 standard limits data to 1492 octets. Some vendors' implementations allow slightly larger transfers.

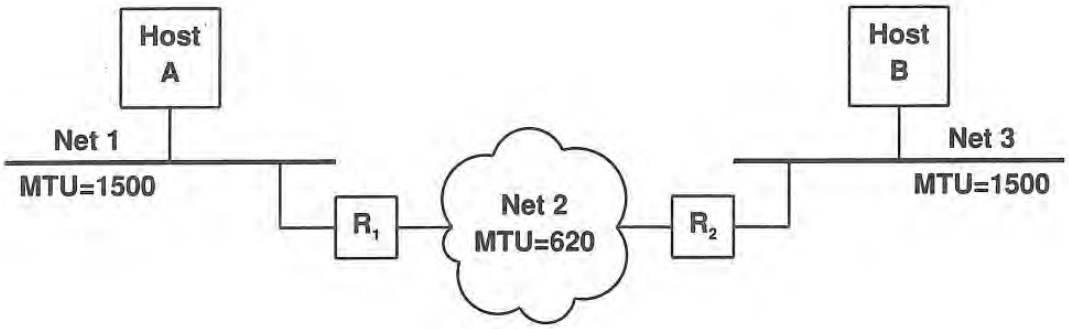

Figure 7.6 An illustration of where fragmentation occurs. Router R_1 fragments large datagrams sent from A to B; R_2 fragments large datagrams sent from B to A.

In the figure, both hosts attach directly to Ethernets which have an MTU of 1500 octets. Thus, both hosts can generate and send datagrams up to 1500 octets long. The path between them, however, includes a network with an MTU of 620. If host A sends host B a datagram larger than 620 octets, router R_1 will fragment the datagram. Similarly, if B sends a large datagram to A, router R_2 will fragment the the datagram.

Fragment size is chosen so each fragment can be shipped across the underlying network in a single frame. In addition, because IP represents the offset of the data in multiples of eight octets, the fragment size must be chosen to be a multiple of eight. Of course, choosing the multiple of eight octets nearest to the network MTU does not usually divide the datagram into equal size pieces; the last piece is often shorter than the others. Fragments must be *reassembled* to produce a complete copy of the original datagram before it can be processed at the destination.

The IP protocol does not limit datagrams to a small size, nor does it guarantee that large datagrams will be delivered without fragmentation. The source can choose any datagram size it thinks appropriate; fragmentation and reassembly occur automatically, without the source taking special action. The IP specification states that routers must accept datagrams up to the maximum of the MTUs of networks to which they attach. In addition, a router must always handle datagrams of up to 576 octets. (Hosts are also required to accept, and reassemble if necessary, datagrams of at least 576 octets.)

Fragmenting a datagram means dividing it into several pieces. It may surprise you to learn that each piece has the same format as the original datagram. Figure 7.7 illustrates the result of fragmentation.

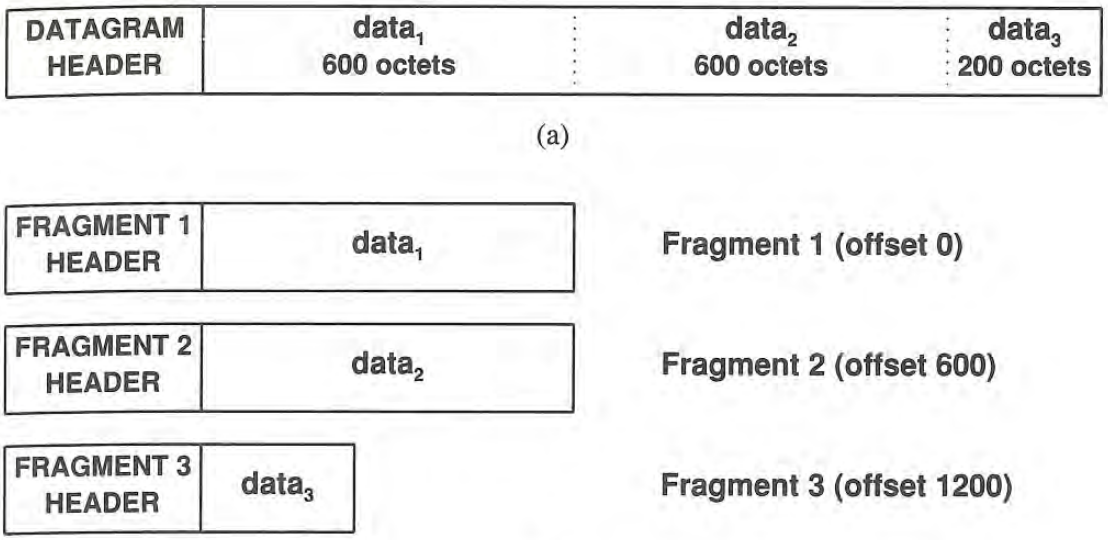

Figure 7.7 (a) An original datagram carrying 1400 octets of data and (b) the three fragments for network MTU of 620. Headers 1 and 2 have the *more fragments* bit set. Offsets shown are decimal octets; they must be divided by 8 to get the value stored in the fragment headers.

Each fragment contains a datagram header that duplicates most of the original datagram header (except for a bit in the *FLAGS* field that shows it is a fragment), followed by as much data as can be carried in the fragment while keeping the total length smaller than the MTU of the network over which it must travel.

7.7.5 Reassembly Of Fragments

Should a datagram be reassembled after passing across one network, or should the fragments be carried to the final host before reassembly? In a TCP/IP internet, once a datagram has been fragmented, the fragments travel as separate datagrams all the way to the ultimate destination where they must be reassembled. Preserving fragments all the way to the ultimate destination has two disadvantages. First, because datagrams are not reassembled immediately after passing across a network with small MTU, the small fragments must be carried from the point of fragmentation to the ultimate destination. Reassembling datagrams at the ultimate destination can lead to inefficiency: even if some of the physical networks encountered after the point of fragmentation have large MTU capability, only small fragments traverse them. Second, if any fragments are lost, the datagram cannot be reassembled. The receiving machine starts a *reassembly timer* when it receives an initial fragment. If the timer expires before all fragments arrive, the

receiving machine discards the surviving pieces without processing the datagram. Thus, the probability of datagram loss increases when fragmentation occurs because the loss of a single fragment results in loss of the entire datagram.

Despite the minor disadvantages, performing reassembly at the ultimate destination works well. It allows each fragment to be routed independently, and does not require intermediate routers to store or reassemble fragments.

7.7.6 Fragmentation Control

Three fields in the datagram header, *IDENTIFICATION*, *FLAGS*, and *FRAGMENT OFFSET*, control fragmentation and reassembly of datagrams. Field *IDENTIFICATION* contains a unique integer that identifies the datagram. Recall that when a router fragments a datagram, it copies most of the fields in the datagram header into each fragment. The *IDENTIFICATION* field must be copied. Its primary purpose is to allow the destination to know which arriving fragments belong to which datagrams. As a fragment arrives, the destination uses the *IDENTIFICATION* field along with the datagram source address to identify the datagram. Computers sending IP datagrams must generate a unique value for the *IDENTIFICATION* field for each unique datagram†. One technique used by IP software keeps a global counter in memory, increments it each time a new datagram is created, and assigns the result as the datagram's *IDENTIFICATION* field.

Recall that each fragment has exactly the same format as a complete datagram. For a fragment, field *FRAGMENT OFFSET* specifies the offset in the original datagram of the data being carried in the fragment, measured in units of 8 octets‡, starting at offset zero. To reassemble the datagram, the destination must obtain all fragments starting with the fragment that has offset *0* through the fragment with highest offset. Fragments do not necessarily arrive in order, and there is no communication between the router that fragmented the datagram and the destination trying to reassemble it.

The low-order two bits of the 3-bit *FLAGS* field controls fragmentation. Usually, application software using TCP/IP does not care about fragmentation because both fragmentation and reassembly are automatic procedures that occur at a low level in the operating system, invisible to end users. However, to test internet software or debug operational problems, it may be important to test sizes of datagrams for which fragmentation occurs. The first control bit aids in such testing by specifying whether the datagram may be fragmented. It is called the *do not fragment* bit because setting it to *1* specifies that the datagram should not be fragmented. An application may choose to disallow fragmentation when only the entire datagram is useful. For example, consider a computer bootstrap sequence in which a machine begins executing a small program in ROM that uses the internet to request an initial bootstrap, and another machine sends back a memory image. If the software has been designed so it needs the entire image or none of it, the datagram should have the *do not fragment* bit set. Whenever a router needs to fragment a datagram that has the *do not fragment* bit set, the router discards the datagram and sends an error message back to the source.

---

†In theory, retransmissions of a datagram can carry the same *IDENTIFICATION* field as the original; in practice, higher-level protocols usually perform retransmission, resulting in a new datagram with its own *IDENTIFICATION*.

‡To save space in the header, offsets are specified in multiples of 8 octets.

The low order bit in the *FLAGS* field specifies whether the fragment contains data from the middle of the original datagram or from the end. It is called the *more fragments* bit. To see why such a bit is needed, consider the IP software at the ultimate destination attempting to reassemble a datagram. It will receive fragments (possibly out of order) and needs to know when it has received all fragments for a datagram. When a fragment arrives, the *TOTAL LENGTH* field in the header refers to the size of the fragment and not to the size of the original datagram, so the destination cannot use the *TOTAL LENGTH* field to tell whether it has collected all fragments. The *more fragments* bit solves the problem easily: once the destination receives a fragment with the *more fragments* bit turned off, it knows this fragment carries data from the tail of the original datagram. From the *FRAGMENT OFFSET* and *TOTAL LENGTH* fields, it can compute the length of the original datagram. By examining the *FRAGMENT OFFSET* and *TOTAL LENGTH* of all fragments that have arrived, a receiver can tell whether the fragments on hand contain all the data needed to reassemble the entire original datagram.

7.7.7 Time to Live (TTL)

Field *TIME TO LIVE* specifies how long, in seconds, the datagram is allowed to remain in the internet system. The idea is both simple and important: whenever a machine injects a datagram into the internet, it sets a maximum time that the datagram should survive. Routers and hosts that process datagrams must decrement the *TIME TO LIVE* field as time passes and remove the datagram from the internet when its time expires.

Estimating exact times is difficult because routers do not usually know the transit time for physical networks. A few rules simplify processing and make it easy to handle datagrams without synchronized clocks. First, each router along the path from source to destination is required to decrement the *TIME TO LIVE* field by *1* when it processes the datagram header. Furthermore, to handle cases of overloaded routers that introduce long delays, each router records the local time when the datagram arrives, and decrements the *TIME TO LIVE* by the number of seconds the datagram remained inside the router waiting for service.

Whenever a *TIME TO LIVE* field reaches zero, the router discards the datagram and sends an error message back to the source. The idea of keeping a timer for datagrams is interesting because it guarantees that datagrams cannot travel around an internet forever, even if routing tables become corrupt and routers route datagrams in a circle.

7.7.8 Other Datagram Header Fields

Field *PROTOCOL* is analogous to the type field in an network frame. The value in the *PROTOCOL* field specifies which high-level protocol was used to create the message being carried in the *DATA* area of a datagram. In essence, the value of *PROTOCOL* specifies the format of the *DATA* area. The mapping between a high level proto-

col and the integer value used in the *PROTOCOL* field must be administered by a central authority to guarantee agreement across the entire Internet.

Field *HEADER CHECKSUM* ensures integrity of header values. The IP checksum is formed by treating the header as a sequence of 16-bit integers (in network byte order), adding them together using one's complement arithmetic, and then taking the one's complement of the result. For purposes of computing the checksum, field *HEADER CHECKSUM* is assumed to contain zero.

It is important to note that the checksum only applies to values in the IP header and not to the data. Separating the checksum for headers and data has advantages and disadvantages. Because the header usually occupies fewer octets than the data, having a separate checksum reduces processing time at routers which only need to compute header checksums. The separation also allows higher level protocols to choose their own checksum scheme for the data. The chief disadvantage is that higher level protocols are forced to add their own checksum or risk having corrupted data go undetected.

Fields *SOURCE IP ADDRESS* and *DESTINATION IP ADDRESS* contain the 32-bit IP addresses of the datagram's sender and intended recipient. Although the datagram may be routed through many intermediate routers, the source and destination fields never change; they specify the IP addresses of the original source and ultimate destination†.

The field labeled *DATA* in Figure 7.3 shows the beginning of the data area of the datagram. Its length depends, of course, on what is being sent in the datagram. The *IP OPTIONS* field, discussed below, is variable length. The field labeled *PADDING*, depends on the options selected. It represents bits containing zero that may be needed to ensure the datagram header extends to an exact multiple of 32 bits (recall that the header length field is specified in units of 32-bit words).

7.8 Internet Datagram Options

The *IP OPTIONS* field following the destination address is not required in every datagram; options are included primarily for network testing or debugging. Options processing is an integral part of the IP protocol, however, so all standard implementations must include it.

The length of the *IP OPTIONS* field varies depending on which options are selected. Some options are one octet long; they consist of a single octet *option code*. Other options are variable length. When options are present in a datagram, they appear contiguously, with no special separators between them. Each option consists of a single octet option code, which may be followed by a single octet length and a set of data octets for that option. The option code octet is divided into three fields as Figure 7.8 shows.

†An exception is made when the datagram includes the source route options listed below.

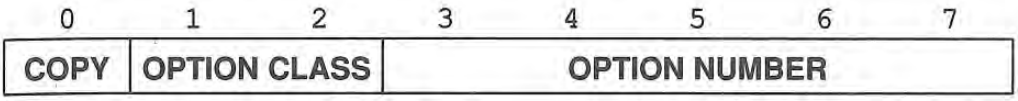

Figure 7.8 The division of the option code octet into three fields of length 1, 2, and 5 bits.

The fields consist of a 1-bit *COPY* flag, a 2-bit *OPTION CLASS*, and the 5-bit *OPTION NUMBER*. The *COPY* flag controls how routers treat options during fragmentation. When the *COPY* bit is set to *1*, it specifies that the option should be copied into all fragments. When set to *0*, the *COPY* bit means that the option should only be copied into the first fragment and not into all fragments.

The *OPTION CLASS* and *OPTION NUMBER* bits specify the general class of the option and give a specific option in that class. The table in Figure 7.9 shows how classes are assigned.

Option Class	Meaning
0	Datagram or network control
1	Reserved for future use
2	Debugging and measurement
3	Reserved for future use

Figure 7.9 Classes of IP options as encoded in the *OPTION CLASS* bits of an option code octet.

The table in Figure 7.10 lists the possible options that can accompany an IP datagram and gives their *OPTION CLASS* and *OPTION NUMBER* values. As the list shows, most options are used for control purposes.

Option Class	Option Number	Length	Description
0	0	-	End of option list. Used if options do not end at end of header (also see header padding field).
0	1	-	No operation (used to align octets in a list of options).
0	2	11	Security and handling restrictions (for military applications).
0	3	var	Loose source routing. Used to route a datagram along a specified path.
0	7	var	Record route. Used to trace a route.
0	8	4	Stream identifier. Used to carry a SATNET stream identifier (Obsolete).
0	9	var	Strict source routing. Used to route a datagram along a specified path.
2	4	var	Internet timestamp. Used to record timestamps along the route.

Figure 7.10 The eight possible IP options with their numeric class and number codes. The value *var* in the length column stands for *variable*.

7.8.1 Record Route Option

The routing and timestamp options are the most interesting because they provide a way to monitor or control how internet routers route datagrams. The *record route* option allows the source to create an empty list of IP addresses and arrange for each router that handles the datagram to add its IP address to the list. Figure 7.11 shows the format of the record route option.

As described above, the *CODE* field contains the option class and option number (*0* and *7* for record route). The *LENGTH* field specifies the total length of the option as it appears in the IP datagram, including the first three octets. The fields starting with one labeled *FIRST IP ADDRESS* comprise the area reserved for recording internet addresses. The *POINTER* field specifies the offset within the option of the next available slot.

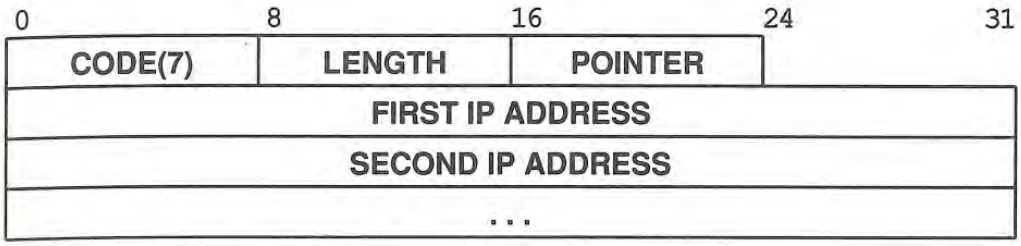

Figure 7.11 The format of the record route option in an IP datagram. The option begins with three octets immediately followed by a list of addresses. Although the diagram shows addresses in 32 bit units, they are not aligned on any octet boundary in a datagram.

Whenever a machine handles a datagram that has the record route option set, the machine adds its address to the record route list (enough space must be allocated in the option by the original source to hold all entries that will be needed). To add itself to the list, a machine first compares the pointer and length fields. If the pointer is greater than the length, the list is full, so the machine forwards the datagram without inserting its entry. If the list is not full, the machine inserts its 4-octet IP address at the position specified by the *POINTER*, and increments the *POINTER* by four.

When the datagram arrives, the destination machine can extract and process the list of IP addresses. Usually, a computer that receives a datagram ignores the recorded route. Using the record route option requires two machines that agree to cooperate; a computer will not automatically receive recorded routes in incoming datagrams after it turns on the record route option in outgoing datagrams. The source must agree to enable the record route option and the destination must agree to process the resultant list.

7.8.2 Source Route Options

Another idea that network builders find interesting is the *source route* option. The idea behind source routing is that it provides a way for the sender to dictate a path through the internet. For example, to test the throughput over a particular physical network, *N*, system administrators can use source routing to force IP datagrams to traverse network *N* even if routers would normally choose a path that did not include it. The ability to make such tests is especially important in a production environment, because it gives the network manager freedom to route users' datagrams over networks that are known to operate correctly while simultaneously testing other networks. Of course, such routing is only useful to people who understand the network topology; the average user has no need to know or use it.

IP supports two forms of source routing. One form, called *strict source routing*, specifies a routing path by including a sequence of IP addresses in the option as Figure 7.12 shows.

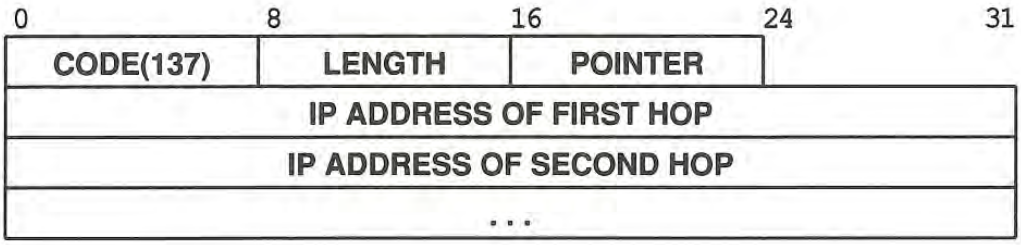

Figure 7.12 The strict source route option specifies an exact route by giving a list of IP addresses the datagram must follow.

Strict source routing means that the addresses specify the exact path the datagram must follow to reach its destination. The path between two successive addresses in the list must consist of a single physical network; an error results if a router cannot follow a strict source route. The other form, called *loose source routing*, also includes a sequence of IP addresses. It specifies that the datagram must follow the sequence of IP addresses, but allows multiple network hops between successive addresses on the list.

Both source route options require routers along the path to overwrite items in the address list with their local network addresses. Thus, when the datagram arrives at its destination, it contains a list of all addresses visited, exactly like the list produced by the record route option.

The format of a source route option resembles that of the record route option shown above. Each router examines the *POINTER* and *LENGTH* fields to see if the list has been exhausted. If it has, the pointer is greater than the length, and the router routes the datagram to its destination as usual. If the list is not exhausted, the router follows the pointer, picks up the IP address, replaces it with the router's address†, and routes the datagram using the address it obtained from the list.

7.8.3 Timestamp Option

The *timestamp option* works like the record route option in that the timestamp option contains an initially empty list, and each router along the path from source to destination fills in one item in the list. Each entry in the list contains two 32-bit items: the IP address of the router that supplied the entry, and a 32-bit integer timestamp. Figure 7.13 shows the format of the timestamp option.

†A router has one address for each interface; it records the address that corresponds to the network over which it routes the datagram.

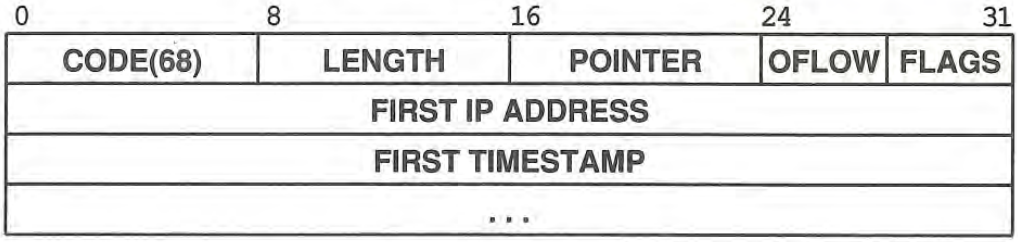

Figure 7.13 The format of the timestamp option. Bits in the FLAGS field control the exact format and rules routers use to process this option.

In the figure, the *LENGTH* and *POINTER* fields are used to specify the length of the space reserved for the option and the location of the next unused slot (exactly as in the record route option). The 4-bit *OFLOW* field contains an integer count of routers that could not supply a timestamp because the option was too small.

The value in the 4-bit *FLAGS* field controls the exact format of the option and tells how routers should supply timestamps. The values are:

Flags value	Meaning
0	Record timestamps only; omit IP addresses.
1	Precede each timestamp by an IP address (this is the format shown in Figure 7.13).
3	IP addresses are specified by sender; a router only records a timestamp if the next IP address in the list matches the router's IP address.

Figure 7.14 The interpretation of values in the FLAGS field of a timestamp option.

Timestamps give the time and date at which a router handles the datagram, expressed as milliseconds since midnight, Universal Time†. If the standard representation for time is unavailable, the router can use any representation of local time provided it turns on the high-order bit in the timestamp field. Of course, timestamps issued by independent computers are not always consistent even if represented in universal time; each machine reports time according to its local clock, and clocks may differ. Thus, timestamp entries should always be treated as estimates, independent of the representation.

It may seem odd that the timestamp option includes a mechanism to have routers record their IP addresses along with timestamps because the record route option already

† Universal Time was formerly called Greenwich Mean Time; it is the time of day at the prime meridian.

provides that capability. However, recording IP addresses with timestamps eliminates ambiguity. Having the route recorded along with timestamps is also useful because it allows the receiver to know exactly which path the datagram followed.

7.8.4 Processing Options During Fragmentation

The idea behind the *COPY* bit in the option *CODE* field should now be clear. When fragmenting a datagram, a router replicates some IP options in all fragments while it places others in only one fragment. For example, consider the option used to record the datagram route. We said that each fragment will be handled as an independent datagram, so there is no guarantee that all fragments follow the same path to the destination. If all fragments contained the record route option, the destination might receive a different list of routes from each fragment. It could not produce a single, meaningful list of routes for the reassembled datagram. Therefore, the IP standard specifies that the record route option should only be copied into one of the fragments.

Not all IP options can be restricted to one fragment. Consider the source route option, for example, that specifies how a datagram should travel through the internet. Source routing information must be replicated in all fragment headers, or fragments will not follow the specified route. Thus, the code field for source route specifies that the option must be copied into all fragments.

7.9 Summary

The fundamental service provided by TCP/IP internet software is a connectionless, unreliable, best-effort packet delivery system. The Internet Protocol (IP) formally specifies the format of internet packets, called *datagrams*, and informally embodies the ideas of connectionless delivery. This chapter concentrated on datagram formats; later chapters will discuss IP routing and error handling.

Analogous to a physical frame, the IP datagram is divided into header and data areas. Among other information, the datagram header contains the source and destination IP addresses, fragmentation control, precedence, and a checksum used to catch transmission errors. Besides fixed-length fields, each datagram header can contain an options field. The options field is variable length, depending on the number and type of options used as well as the size of the data area allocated for each option. Intended to help monitor and control the internet, options allow one to specify or record routing information, or to gather timestamps as the datagram traverses an internet.

FOR FURTHER STUDY

Postel [1980] discusses possible ways to approach internet protocols, addressing, and routing. In later publications, Postel [RFC 791] gives the standard for the Internet Protocol. Braden [RFC 1122] further refines the standard. Hornig [RFC 894] specifies the standard for the transmission of IP datagrams across an Ethernet. Clark [RFC 815] describes efficient reassembly of fragments. In addition to the packet format, Internet authorities also specify many constants needed in the network protocols. These values can be found in Reynolds and Postel [RFC 1700]. Kent and Mogul [1987] discusses the disadvantages of fragmentation.

An alternative internet protocol suite known as *XNS*, is given in Xerox [1981]. Boggs *et. al.* [1980] describes the PARC Universal Packet (PUP) protocol, an abstraction from *XNS* closely related to the IP datagram.

EXERCISES

7.1 What is the single greatest advantage of having the IP checksum cover only the datagram header and not the data? What is the disadvantage?

7.2 Is it ever necessary to use an IP checksum when sending packets over an Ethernet? Why or why not?

7.3 What is the MTU size for the ANSNET? Hyperchannel? an ATM switch?

7.4 Do you expect a high-speed local area network to have larger or smaller MTU size than a wide area network?

7.5 Argue that fragments should have small, nonstandard headers.

7.6 Find out when the IP protocol version last changed. Is having a protocol version number really useful?

7.7 Can you imagine why a one's complement checksum was chosen for IP instead of a cyclic redundancy check?

7.8 What are the advantages of doing reassembly at the ultimate destination instead of doing it after the datagram travels across one network?

7.9 What is the minimum network MTU required to send an IP datagram that contains at least one octet of data?

7.10 Suppose you are hired to implement IP datagram processing in hardware. Is there any rearrangement of fields in the header that would have made your hardware more efficient? Easier to build?

7.11 If you have access to an implementation of IP, revise it and test your locally available implementations of IP to see if they reject IP datagrams with an out-of-date version number.

7.12 When a minimum-size IP datagram travels across an Ethernet, how large is the frame?

11

Protocol Layering

11.1 Introduction

Previous chapters review the architectural foundations of internetworking, describe how hosts and routers forward Internet datagrams, and present mechanisms used to map IP addresses to physical network addresses. This chapter considers the structure of the software found in hosts and routers that carries out network communication. It presents the general principle of layering, shows how layering makes Internet Protocol software easier to understand and build, and traces the path of datagrams through the protocol software they encounter when traversing a TCP/IP internet.

11.2 The Need For Multiple Protocols

We have said that protocols allow one to specify or understand communication without knowing the details of a particular vendor's network hardware. They are to computer communication what programming languages are to computation. It should be apparent by now how closely the analogy fits. Like assembly language, some protocols describe communication across a physical network. For example, the details of the Ethernet frame format, network access policy, and frame error handling comprise a protocol that describes communication on an Ethernet. Similarly, the details of IP addresses, the datagram format, and the concept of unreliable, connectionless delivery comprise the Internet Protocol.

Complex data communication systems do not use a single protocol to handle all transmission tasks. Instead, they require a set of cooperative protocols, sometimes called a *protocol family* or *protocol suite*. To understand why, think of the problems that arise when machines communicate over a data network:

• *Hardware failure.* A host or router may fail either because the hardware fails or because the operating system crashes. A network transmission link may fail or accidentally be disconnected. The protocol software needs to detect such failures and recover from them if possible.

• *Network congestion.* Even when all hardware and software operates correctly, networks have finite capacity that can be exceeded. The protocol software needs to arrange ways that a congested machine can suppress further traffic.

• *Packet delay or loss.* Sometimes, packets experience extremely long delays or are lost. The protocol software needs to learn about failures or adapt to long delays.

• *Data corruption.* Electrical or magnetic interference or hardware failures can cause transmission errors that corrupt the contents of transmitted data. Protocol software needs to detect and recover from such errors.

• *Data duplication or sequence errors.* Networks that offer multiple routes may deliver data out of sequence or may deliver duplicates of packets. The protocol software needs to reorder packets and remove any duplicates.

Taken together, all these problems seem overwhelming. It is difficult to understand how to write a single protocol that will handle them all. From the analogy with programming languages, we can see how to conquer the complexity. Program translation has been partitioned into four conceptual subproblems identified with the software that handles each subproblem: compiler, assembler, link editor, and loader. The division makes it possible for the designer to concentrate on one subproblem at a time, and for the implementor to build and test each piece of software independently. We will see that protocol software is partitioned similarly.

Two final observations about our programming language analogy will help clarify the organization of protocols. First, it should be clear that pieces of translation software must agree on the exact format of data passed between them. For example, the data passed from the compiler to the assembler consists of a program defined by the assembly programming language. Thus, we see how the translation process involves multiple programming languages. The analogy will hold for communication software, where we will see that multiple protocols define the interfaces between the modules of communication software. Second, the four parts of the translator form a linear sequence in which output from the compiler becomes input to the assembler, and so on. Protocol software also uses a linear sequence.

11.3 The Conceptual Layers Of Protocol Software

Think of the modules of protocol software on each machine as being stacked vertically into *layers*, as in Figure 11.1. Each layer takes responsibility for handling one part of the problem.

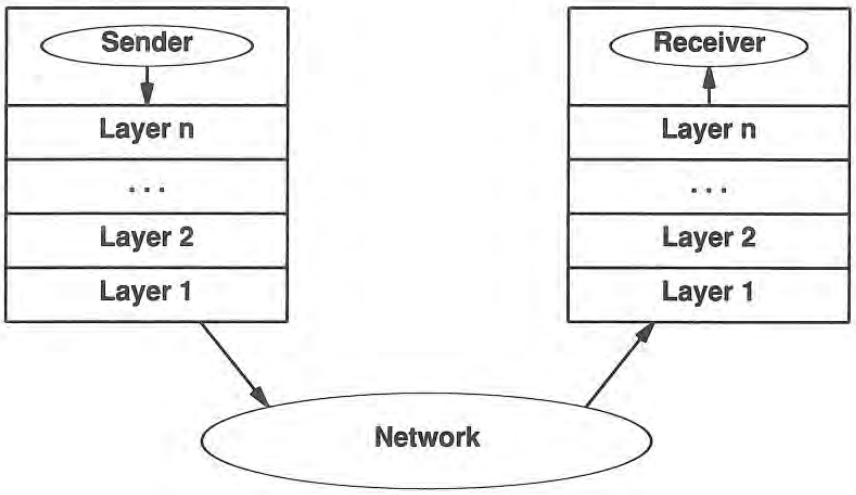

Figure 11.1 The conceptual organization of protocol software in layers.

Conceptually, sending a message from an application program on one machine to an application program on another means transferring the message down through successive layers of protocol software on the sender's machine, transferring the message across the network, and transferring the message up through successive layers of protocol software on the receiver's machine.

In practice, the protocol software is much more complex than the simple model of Figure 11.1 indicates. Each layer makes decisions about the correctness of the message and chooses an appropriate action based on the message type or destination address. For example, one layer on the receiving machine must decide whether to keep the message or forward it to another machine. Another layer must decide which application program should receive the message.

To understand the difference between the conceptual organization of protocol software and the implementation details, consider the comparison shown in Figure 11.2. The conceptual diagram in Figure 11.2a shows an Internet layer between a high level protocol layer and a network interface layer. The realistic diagram in Figure 11.2b shows that the IP software may communicate with multiple high-level protocol modules and with multiple network interfaces.

Although a diagram of conceptual protocol layering does not show all details, it does help explain the general ideas. For example, Figure 11.3 shows the layers of protocol software used by a message that traverses three networks. The diagram shows only the network interface and Internet Protocol layers in routers because only those layers are needed to receive, route, and then send datagrams. We understand that any machine attached to two networks must have two network interface modules, even though the conceptual layering diagram shows only a single network interface layer in each machine.

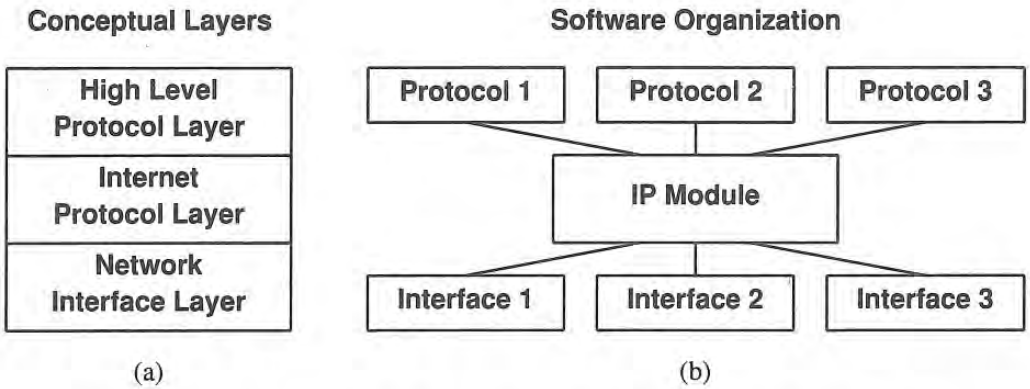

Figure 11.2 A comparison of (a) conceptual protocol layering and (b) a realistic view of software organization showing multiple network interfaces below IP and multiple protocols above it.

As Figure 11.3 shows, a sender on the original machine transmits a message which the IP layer places in a datagram and sends across network *1*. On intermediate machines the datagram passes up to the IP layer which routes it back out again (on a different network). Only when it reaches the final destination machine does IP extract the message and pass it up to higher layers of protocol software.

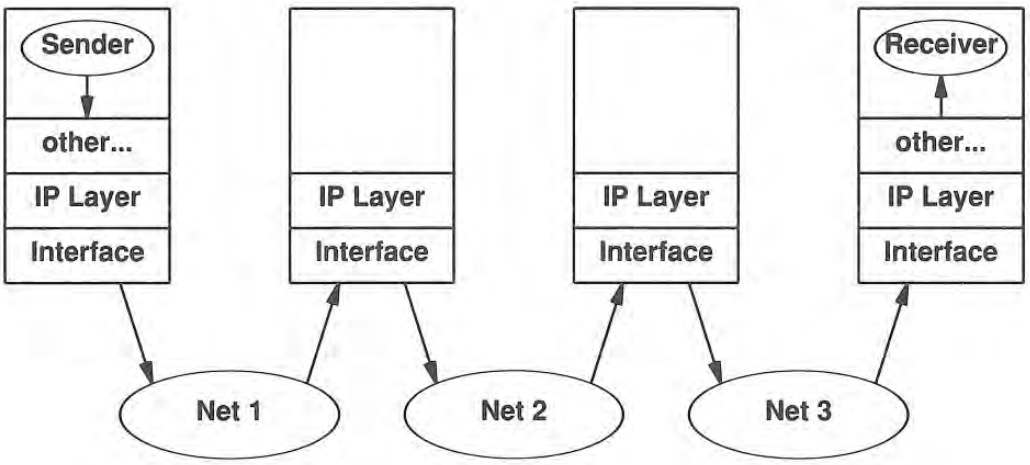

Figure 11.3 The path of a message traversing the Internet from the sender through two intermediate machines to the receiver. Intermediate machines only send the datagram to the IP software layer.

11.4 Functionality Of The Layers

Once the decision has been made to partition the communication problem into subproblems and organize the protocol software into modules that each handle one subproblem, the question arises: ''what functionality should reside in each module?'' The question is not easy to answer for several reasons. First, given a set of goals and constraints governing a particular communication problem, it is possible to choose an organization that will optimize protocol software for that problem. Second, even when considering general network-level services such as reliable transport, it is possible to choose from among fundamentally distinct approaches to solving the problem. Third, the design of network (or internet) architecture and the organization of the protocol software are interrelated; one cannot be designed without the other.

11.4.1 ISO 7-Layer Reference Model

Two ideas about protocol layering dominate the field. The first, based on work done by the International Organization for Standardization (ISO), is known as ISO's *Reference Model of Open System Interconnection*, often referred to as the *ISO model*. The ISO model contains 7 conceptual layers organized as Figure 11.4 shows.

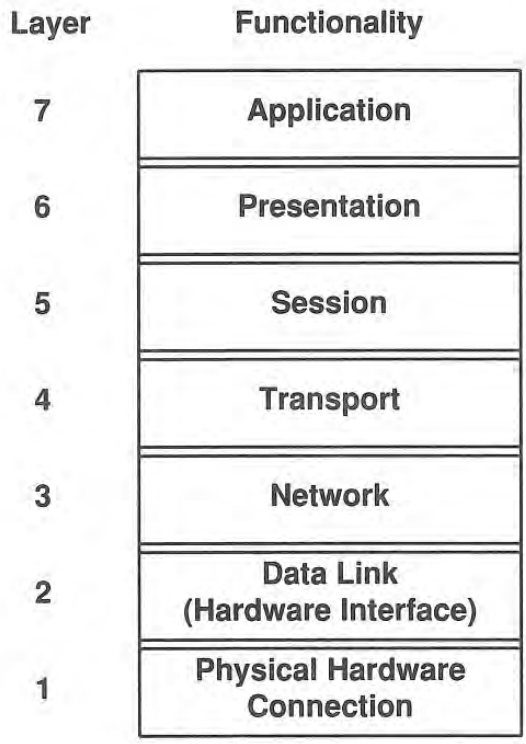

Figure 11.4 The ISO 7-layer reference model for protocol software.

The ISO model, built to describe protocols for a single network, does not contain a specific level for internetwork routing in the same way TCP/IP protocols do.

11.5 X.25 And Its Relation To The ISO Model

Although it was designed to provide a conceptual model and not an implementation guide, the ISO layering scheme has been the basis for several protocol implementations. Among the protocols commonly associated with the ISO model, the set of protocols known as X.25 is probably the best known and most widely used. X.25 was established as a recommendation of the *Telecommunications Section* of the *International Telecommunications Union*† (ITU-TS), an international organization that recommends standards for international telephone services. X.25 has been adopted by public data networks, and is especially popular in Europe. Considering X.25 will help explain the ISO layering.

In the X.25 view, a network operates much like a telephone system. An X.25 network is assumed to consist of complex packet switches that contain the intelligence needed to route packets. Hosts do not attach directly to communication wires of the network. Instead each host attaches to one of the packet switches using a serial communication line. In one sense the connection between a host and an X.25 packet switch is a miniature network consisting of one serial link. The host must follow a complicated procedure to transfer packets onto the network.

• *Physical Layer.* X.25 specifies a standard for the physical interconnection between host computers and network packet switches, as well as the procedures used to transfer packets from one machine to another. In the reference model, level *1* specifies the physical interconnection including electrical characteristics of voltage and current. A corresponding protocol, X.21, gives the details used by public data networks.

• *Data Link Layer.* The level *2* portion of the X.25 protocol specifies how data travels between a host and the packet switch to which it connects. X.25 uses the term *frame* to refer to a unit of data as it passes between a host and a packet switch (it is important to understand that the X.25 definition of *frame* differs slightly from the way we have used it). Because raw hardware delivers only a stream of bits, the level *2* protocol must define the format of frames and specify how the two machines recognize frame boundaries. Because transmission errors can destroy data, the level *2* protocol includes error detection (e.g., a frame checksum). Finally, because transmission is unreliable, the level *2* protocol specifies an exchange of acknowledgements that allows the two machines to know when a frame has been transferred successfully.

One commonly used level *2* protocol, named the *High Level Data Link Communication*, is best known by its acronym, *HDLC*. Several versions of HDLC exist, with the most recent known as *HDLC/LAPB*. It is important to remember that successful transfer at level *2* means a frame has been passed to the network packet switch for delivery; it does not guarantee that the packet switch accepted the packet or was able to route it.

---

†The International Telecommunications Union was formerly named the *Consultative Committee on International Telephony and Telegraphy* (CCITT).

• *Network Layer.* The ISO reference model specifies that the third level contains functionality that completes the definition of the interaction between host and network. Called the *network* or *communication subnet* layer, this level defines the basic unit of transfer across the network and includes the concepts of destination addressing and routing. Remember that in the X.25 world, communication between host and packet switch is conceptually isolated from the traffic that is being passed. Thus, the network might allow packets defined by level 3 protocols to be larger than the size of frames that can be transferred at level 2. The level 3 software assembles a packet in the form the network expects and uses level 2 to transfer it (possibly in pieces) to the packet switch. Level 3 must also respond to network congestion problems.

• *Transport Layer.* Level 4 provides end-to-end reliability by having the destination host communicate with the source host. The idea here is that even though lower layers of protocols provide reliable checks at each transfer, the end-to-end layer double checks to make sure that no machine in the middle failed.

• *Session Layer.* Higher levels of the ISO model describe how protocol software can be organized to handle all the functionality needed by application programs. The ISO committee considered the problem of remote terminal access so fundamental that they assigned layer 5 to handle it. In fact, the central service offered by early public data networks consisted of terminal to host interconnection. The carrier provides a special purpose host computer called a *Packet Assembler And Disassembler* (*PAD*) on the network with dialup access. Subscribers, often travelers who carry their own computer and modem, dial up the local PAD, make a network connection to the host with which they wish to communicate, and log in. Many carriers choose to make using the network for long distance communication less expensive than direct dialup.

• *Presentation Layer.* ISO layer 6 is intended to include functions that many application programs need when using the network. Typical examples include standard routines that compress text or convert graphics images into bit streams for transmission across a network. For example an ISO standard known as *Abstract Syntax Notation 1* (*ASN.1*), provides a representation of data that application programs use. One of the TCP/IP protocols, SNMP, also uses ASN.1 to represent data.

• *Application Layer.* Finally, ISO layer 7 includes application programs that use the network. Examples include electronic mail or file transfer programs. In particular, the ITU-TS has devised a protocol for electronic mail known as the *X.400* standard. In fact, the ITU and ISO worked jointly on message handling systems; the ISO version is called *MOTIS*.

11.5.1 The TCP/IP Internet Layering Model

The second major layering model did not arise from a standards committee, but came instead from research that led to the TCP/IP protocol suite. With a little work, the ISO model can be stretched to describe the TCP/IP layering scheme, but the underlying assumptions are different enough to warrant distinguishing the two.

Broadly speaking, TCP/IP software is organized into four conceptual layers that build on a fifth layer of hardware. Figure 11.5 shows the conceptual layers as well as the form of data as it passes between them.

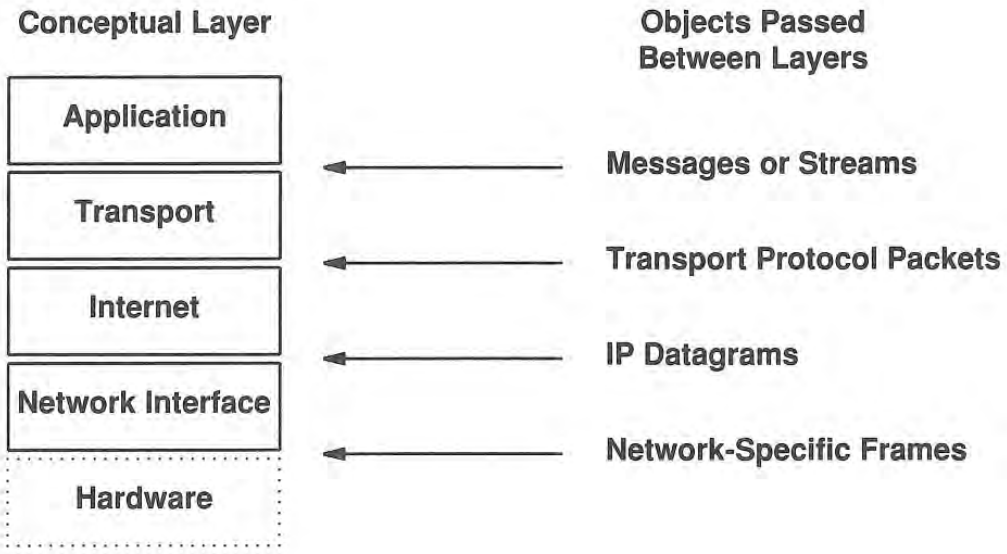

Figure 11.5 The 4 conceptual layers of TCP/IP software and the form of objects passed between layers. The layer labeled *network interface* is sometimes called the *data link* layer.

• *Application Layer.* At the highest level, users invoke application programs that access services available across a TCP/IP internet. An application interacts with one of the transport level protocols to send or receive data. Each application program chooses the style of transport needed, which can be either a sequence of individual messages or a continuous stream of bytes. The application program passes data in the required form to the transport level for delivery.

• *Transport Layer.* The primary duty of the *transport layer* is to provide communication from one application program to another. Such communication is often called *end-to-end.* The transport layer may regulate flow of information. It may also provide reliable transport, ensuring that data arrives without error and in sequence. To do so, transport protocol software arranges to have the receiving side send back acknowledgements and the sending side retransmit lost packets. The transport software divides the stream of data being transmitted into small pieces (sometimes called *packets*) and passes each packet along with a destination address to the next layer for transmission.

Although Figure 11.5 uses a single block to represent the application layer, a general purpose computer can have multiple application programs accessing the internet at one time. The transport layer must accept data from several user programs and send it

to the next lower layer. To do so, it adds additional information to each packet, including codes that identify which application program sent it and which application program should receive it, as well as a checksum. The receiving machine uses the checksum to verify that the packet arrived intact, and uses the destination code to identify the application program to which it should be delivered.

• *Internet Layer.* As we have already seen, the Internet layer handles communication from one machine to another. It accepts a request to send a packet from the transport layer along with an identification of the machine to which the packet should be sent. It encapsulates the packet in an IP datagram, fills in the datagram header, uses the routing algorithm to determine whether to deliver the datagram directly or send it to a router, and passes the datagram to the appropriate network interface for transmission. The Internet layer also handles incoming datagrams, checking their validity, and uses the routing algorithm to decide whether the datagram should be processed locally or forwarded. For datagrams addressed to the local machine, software in the internet layer deletes the datagram header, and chooses from among several transport protocols the one that will handle the packet. Finally, the Internet layer sends ICMP error and control messages as needed and handles all incoming ICMP messages.

• *Network Interface Layer.* The lowest level TCP/IP software comprises a network interface layer, responsible for accepting IP datagrams and transmitting them over a specific network. A network interface may consist of a device driver (e.g., when the network is a local area network to which the machine attaches directly) or a complex subsystem that uses its own data link protocol (e.g., when the network consists of packet switches that communicate with hosts using HDLC).

11.6 Differences Between X.25 And Internet Layering

There are two subtle and important differences between the TCP/IP layering scheme and the X.25 scheme. The first difference revolves around the focus of attention on reliability, while the second involves the location of intelligence in the overall system.

11.6.1 Link-Level vs. End-To-End Reliability

One major difference between the TCP/IP protocols and the X.25 protocols lies in their approaches to providing reliable data transfer services. In the X.25 model, protocol software detects and handles errors at all levels. At the link level, complex protocols guarantee that the transfer between a host and the packet switch to which it connects will be correct. Checksums accompany each piece of data transferred, and the receiver acknowledges each piece of data received. The link level protocol includes timeout and retransmission algorithms that prevent data loss and provide automatic recovery after hardware fails and restarts.

Successive levels of X.25 provide reliability of their own. At level *3*, X.25 also provides error detection and recovery for packets transferred onto the network, using checksums as well as timeout and retransmission techniques. Finally, level *4* must provide end-to-end reliability, having the source correspond with the ultimate destination to verify delivery.

In contrast to such a scheme, TCP/IP bases its protocol layering on the idea that reliability is an end-to-end problem. The architectural philosophy is simple: construct the internet so it can handle the expected load, but allow individual links or machines to lose data or corrupt it without trying to repeatedly recover. In fact, there is little or no reliability in most TCP/IP network interface layer software. Instead, the transport layer handles most error detection and recovery problems.

The resulting freedom from interface layer verification makes TCP/IP software much easier to understand and implement correctly. Intermediate routers can discard datagrams that become corrupted because of transmission errors. They can discard datagrams that cannot be delivered. They can discard datagrams when the arrival rate exceeds machine capacity, and can reroute datagrams through paths with shorter or longer delay without informing the source or destination.

Having unreliable links means that some datagrams do not arrive. Detection and recovery of datagram loss is carried out between the source host and the ultimate destination and is, therefore, called *end-to-end* verification. The end-to-end software located in the transport layer uses checksums, acknowledgements, and timeouts to control transmission. Thus, unlike the connection-oriented X.25 protocol layering, the TCP/IP software focuses most of its reliability control in one layer.

11.6.2 Locus of Intelligence and Decision Making

Another difference between the X.25 model and the TCP/IP model emerges when one considers the locus of authority and control. As a general rule, networks using X.25 adhere to the idea that a network is a utility that provides a transport service. The vendor that offers the service controls network access and monitors traffic to keep records for accounting and billing. The network vendor also handles problems like routing, flow control, and acknowledgements internally, making transfers reliable. This view leaves little that the hosts can (or need to) do. In short, the network is a complex, independent system to which one can attach relatively simple host computers; the hosts themselves participate in the network operation very little.

By contrast, TCP/IP requires hosts to participate in almost all of the network protocols. We have already mentioned that hosts actively implement end-to-end error detection and recovery. They also participate in routing because they must choose a router when sending datagrams, and they participate in network control because they must handle ICMP control messages. Thus, when compared to an X.25 network, a TCP/IP internet can be viewed as a relatively simple packet delivery system to which intelligent hosts attach.

11.7 The Protocol Layering Principle

Independent of the particular layering scheme used, or the functions of the layers, the operation of layered protocols is based on a fundamental idea. The idea, called the *layering principle*, can be summarized succinctly:

> *Layered protocols are designed so that layer* n *at the destination receives exactly the same object sent by layer* n *at the source.*

The layering principle explains why layering is such a powerful idea. It allows the protocol designer to focus attention on one layer at a time, without worrying about how lower layers perform. For example, when building a file transfer application, the designer thinks only of two copies of the application program executing on two machines and concentrates on the messages they need to exchange for file transfer. The designer assumes that the application on one host receives exactly what the application on the other host sends.

Figure 11.6 illustrates how the layering principle works:

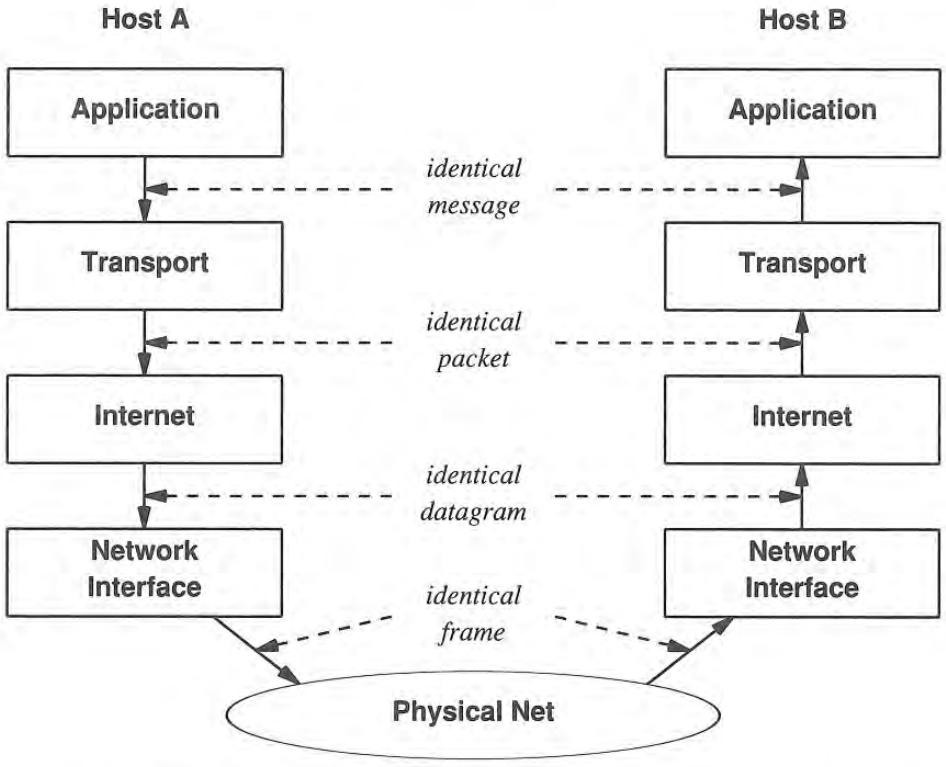

Figure 11.6 The path of a message as it passes from an application on one host to an application on another. Layer *n* on host *B* receives exactly the same object that layer *n* on host *A* sent.

11.7.1 Layering in a TCP/IP Internet Environment

Our statement of the layering principle is somewhat vague, and the illustration in Figure 11.6 skims over an important issue because it fails to distinguish between transfers from source to ultimate destination and transfers across multiple networks. Figure 11.7 illustrates the distinction, showing the path of a message sent from an application program on one host to an application on another through a router.

As the figure shows, message delivery uses two separate network frames, one for the transmission from host *A* to router *R*, and another from router *R* to host *B*. The network layering principle states that the frame delivered to *R* is identical to the frame sent by host *A*. By contrast, the application and transport layers deal with end-to-end issues and are designed so the software at the source communicates with its peer at the ultimate destination. Thus, the layering principle states that the packet received by the transport layer at the ultimate destination is identical to the packet sent by the transport layer at the original source.

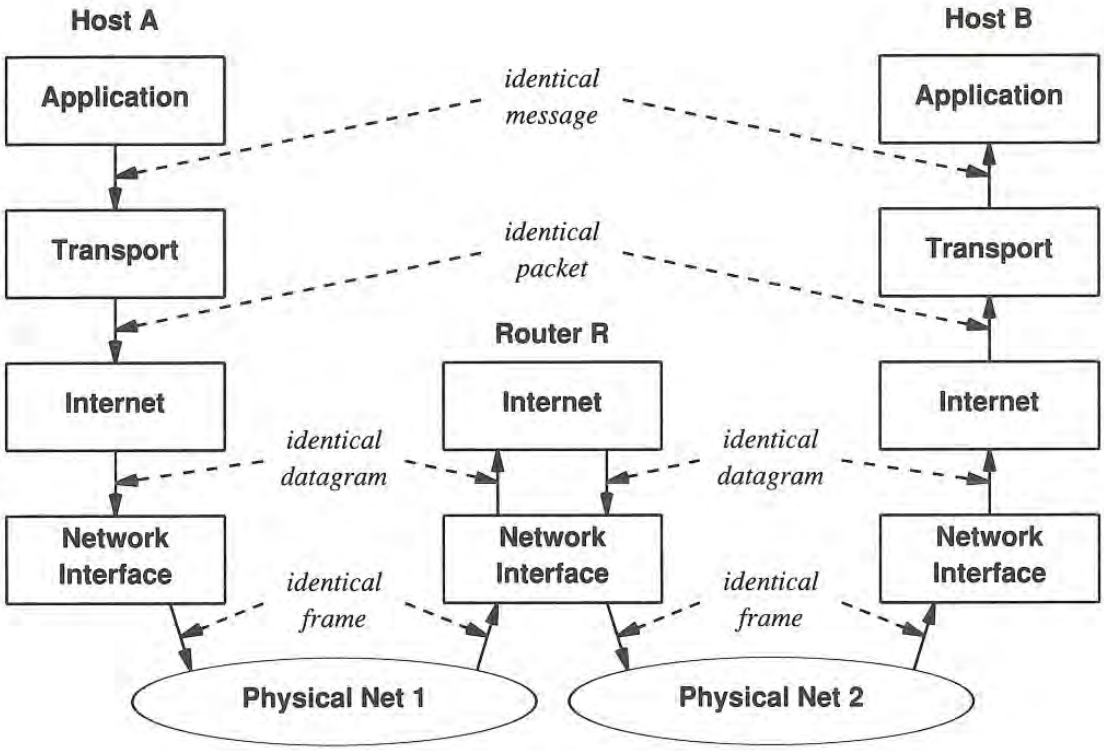

Figure 11.7 The layering principle when a router is used. The frame delivered to router *R* is exactly the frame sent from host *A*, but differs from the frame sent between *R* and *B*.

It is easy to understand that in higher layers, the layering principle applies across end-to-end transfers, and that at the lowest layer it applies to a single machine transfer. It is not as easy to see how the layering principle applies to the Internet layer. On one hand, we have said that hosts attached to an internet should view it as a large, virtual network, with the IP datagram taking the place of a network frame. In this view, datagrams travel from original source to ultimate destination, and the layering principle guarantees that the ultimate destination receives exactly the datagram that the original source sent. On the other hand, we know that the datagram header contains fields, like a *time to live* counter, that change each time the datagram passes through a router. Thus, the ultimate destination will not receive exactly the same datagram as the source sent. We conclude that although most of the datagram stays intact as it passes across an internet, the layering principle only applies to datagrams across single machine transfers. To be accurate, we should not view the Internet layer as providing end-to-end service.

11.8 Layering In The Presence Of Network Substructure

Recall from Chapter 2 that some wide area networks contain multiple packet switches. For example, a WAN can consist of routers that connect to a local network at each site as well as to other routers using leased serial lines. When a router receives a datagram, it either delivers the datagram to its destination on the local network, or transfers the datagram across a serial line to another router. The question arises: ''How do the protocols used on serial lines fit into the TCP/IP layering scheme?'' The answer depends on how the designer views the serial line interconnections.

From the perspective of IP, the set of point-to-point connections among routers can either function like a set of independent physical networks, or they can function collectively like a single physical network. In the first case, each physical link is treated exactly like any other network in the internet. It is assigned a unique (usually class C) network number, and the two hosts that share the link each have a unique IP address assigned for their connection. Routes are added to the IP routing table as they would be for any other network. A new software module is added at the network interface layer to control the new link hardware, but no substantial changes are made to the layering scheme. The main disadvantage of the independent network approach is that it proliferates network numbers (one for each connection between two machines), causing routing tables to be larger than necessary. Both *Serial Line IP* (*SLIP*) and the *Point to Point Protocol* (*PPP*) treat each serial link as a separate network.

The second approach to accommodating point-to-point connections avoids assigning multiple IP addresses to the physical wires. Instead, it treats all the connections collectively as a single, independent IP network with its own frame format, hardware addressing scheme, and data link protocols. Routers that use the second approach need only one IP network number for all point-to-point connections.

Using the single network approach means extending the protocol layering scheme to add a new intranetwork routing layer between the network interface layer and the hardware devices. For machines with only one point-to-point connection, an additional layer seems unnecessary. To see why it is needed, consider a machine with several physical point-to-point connections, and recall from Figure 11.2 how the network interface layer is divided into multiple software modules that each control one network. We need to add one new network interface for the new point-to-point network, but the new interface must control multiple hardware devices. Furthermore, given a datagram to send, the new interface must choose the correct link over which the datagram should be sent. Figure 11.8 shows the organization.

The Internet layer software passes to the network interface all datagrams that should be sent out on any of the point-to-point connections. The interface passes them to the intranet routing module that must further distinguish among multiple physical connections and route the datagram across the correct one.

The programmer who designs the intranet routing software determines exactly how the software chooses a physical link. Usually, the algorithm relies on an intranet routing table. The intranet routing table is analogous to the internet routing table in that it specifies a mapping of destination address to route. The table contains pairs of entries, (D, L), where D is a destination host address and L specifies one of the physical lines used to reach that destination.

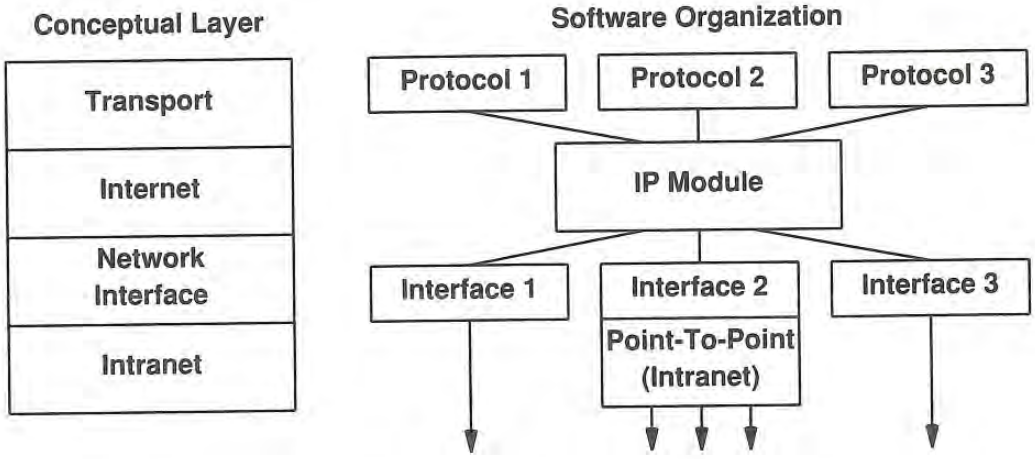

Figure 11.8 (a) conceptual position of an intranet protocol for point-to-point connections when IP treats them as a single IP network, and (b) detailed diagram of corresponding software modules. Each arrow corresponds to one physical device.

The difference between an internet routing table and an intranet routing table is that intranet routing tables are quite small. They only contain routing information for hosts directly attached to the point-to-point network. The reason is simple: the Internet layer maps an arbitrary destination address to a specific router address before passing the datagram to a network interface. Thus, the intranet layer is asked only to distinguish among machines on a single point-to-point network.

11.9 Two Important Boundaries In The TCP/IP Model

The conceptual protocol layering includes two boundaries that may not be obvious: a protocol address boundary that separates high-level and low-level addressing, and an operating system boundary that separates the system from application programs.

11.9.1 High-Level Protocol Address Boundary

Now that we have seen the layering of TCP/IP software, we can be precise about an idea introduced in Chapter 8: a conceptual boundary partitions software that uses low-level (physical) addresses from software that uses high-level (IP) addresses. As Figure 11.9 shows, the boundary occurs between the network interface layer and the Internet layer. That is,

> *Application programs as well as all protocol software from the Internet layer upward use only IP addresses; the network interface layer handles physical addresses.*

Thus, protocols like ARP belong in the network interface layer. They are not part of IP.

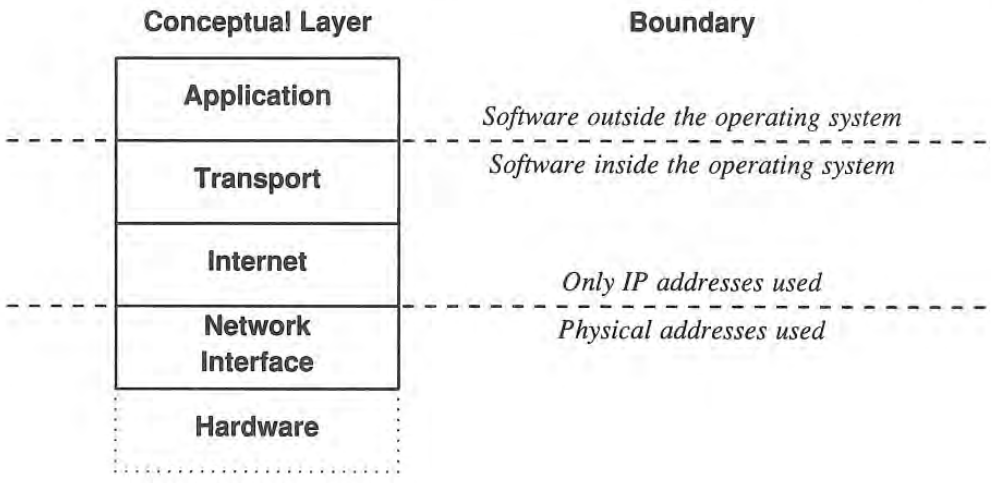

Figure 11.9 The relationship between conceptual layering and the boundaries for operating system and high-level protocol addresses.

11.9.2 Operating System Boundary

Figure 11.9 shows another important boundary as well, the division between software that is generally considered part of the operating system and software that is not. While each implementation of TCP/IP chooses how to make the distinction, many follow the scheme shown. Because they lie inside the operating system, passing data between lower layers of protocol software is much less expensive than passing it between an application program and a transport layer. Chapter 20 discusses the problem in more detail and describes an example of the interface an operating system might provide.

11.10 The Disadvantage Of Layering

We have said that layering is a fundamental idea that provides the basis for protocol design. It allows the designer to divide a complicated problem into subproblems and solve each one independently. Unfortunately, the software that results from strict layering can be extremely inefficient. As an example, consider the job of the transport layer. It must accept a stream of bytes from an application program, divide the stream into packets, and send each packet across the internet. To optimize transfer, the transport layer should choose the largest possible packet size that will allow one packet to travel in one network frame. In particular, if the destination machine attaches directly to one of the same networks as the source, only one physical net will be involved in the transfer, so the sender can optimize packet size for that network. If the software preserves strict layering, however, the transport layer cannot know how the Internet module will route traffic or which networks attach directly. Furthermore, the transport layer will not understand the datagram or frame formats nor will it be able to determine how many octets of header will be added to a packet. Thus, strict layering will prevent the transport layer from optimizing transfers.

Usually, implementors relax the strict layering scheme when building protocol software. They allow information like route selection and network MTU to propagate upward. When allocating buffers, they often leave space for headers that will be added by lower layer protocols and may retain headers on incoming frames when passing them to higher layer protocols. Such optimizations can make dramatic improvements in efficiency while retaining the basic layered structure.

11.11 The Basic Idea Behind Multiplexing And Demultiplexing

Communication protocols uses techniques of *multiplexing* and *demultiplexing* throughout the layered hierarchy. When sending a message, the source computer includes extra bits that encode the message type, originating program, and protocols used.

Eventually, all messages are placed into network frames for transfer and combined into a stream of packets. At the receiving end, the destination machine uses the extra information to guide processing.

Consider an example of demultiplexing shown in Figure 11.10.

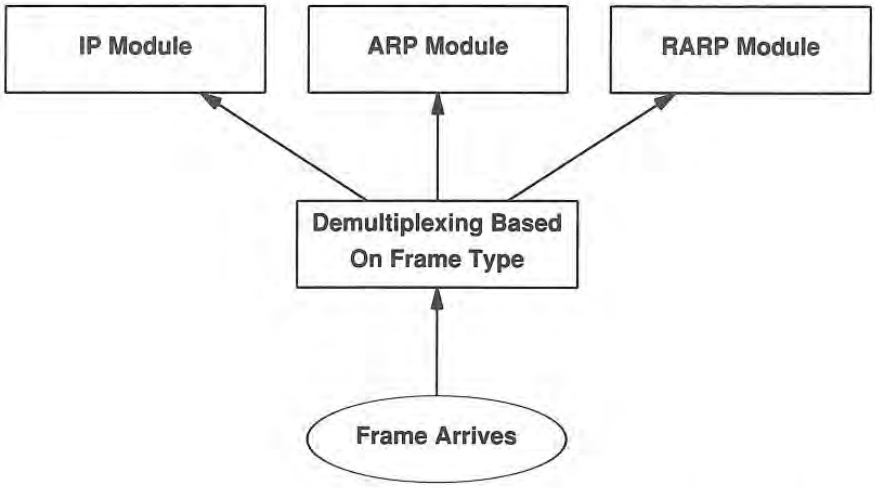

Figure 11.10 Demultiplexing of incoming frames based on the type field-found in the frame header.

The figure illustrates how software in the network interface layer uses the frame type to choose a procedure that handles the incoming frame. We say that the network interface *demultiplexes* the frame based on its type. To make such a choice possible, software in the source machine must set the frame type field before transmission. Thus, each software module that sends frames uses the type field to specify frame contents.

Multiplexing and demultiplexing occur at almost every protocol layer. For example, after the network interface demultiplexes frames and passes those frames that contain IP datagrams to the IP module, the IP software extracts the datagram and demultiplexes further based on the transport protocol. Figure 11.11 demonstrates demultiplexing at the Internet layer.

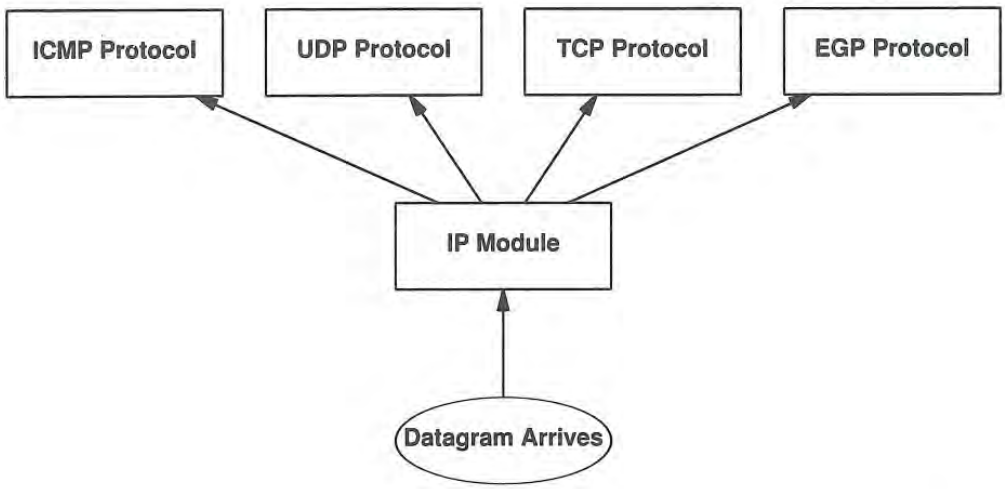

Figure 11.11 Demultiplexing at the Internet layer. IP software chooses an appropriate procedure to handle a datagram based on the protocol type field in the datagram header.

To decide how to handle a datagram, internet software examines the header of a datagram and selects a protocol handler based on the datagram type. In the example, the possible datagram types are: *ICMP*, which we have already examined, and *UDP*, *TCP*, and *EGP*, which we will examine in later chapters.

11.12 Summary

Protocols are the standards that specify how data is represented when being transferred from one machine to another. Protocols specify how the transfer occurs, how errors are detected, and how acknowledgements are passed. To simplify protocol design and implementation, communication problems are segregated into subproblems that can be solved independently. Each subproblem is assigned a separate protocol.

The idea of layering is fundamental because it provides a conceptual framework for protocol design. In a layered model, each layer handles one part of the communication problem and usually corresponds to one protocol. Protocols follow the layering principle, which states that the software implementing layer n on the destination machine receives exactly what the software implementing layer n on the source machine sends.

We examined the 4-layer Internet reference model as well as the ISO 7-layer reference model. In both cases, the layering model provides only a conceptual framework for protocol software. The ITU-TS X.25 protocols follow the ISO reference model and provide an example of reliable communication service offered by a commercial utility, while the TCP/IP protocols provide an example of a different layering scheme.

In practice, protocol software uses multiplexing and demultiplexing to distinguish among multiple protocols within a given layer, making protocol software more complex than the layering model suggests.

FOR FURTHER STUDY

Postel [RFC 791] provides a sketch of the Internet Protocol layering scheme, and Clark [RFC 817] discusses the effect of layering on implementations. Saltzer, Reed, and Clark [1984] argues that end-to-end verification is important. Chesson [1987] makes the controversial argument that layering produces intolerably bad network throughput. Volume 2 of this text examines layering in detail, and shows an example implementation that achieves efficiency by compromising strict layering and passing pointers between layers.

The ISO protocol documents [1987a] and [1987b] describe ASN.1 in detail. Sun [RFC 1014] describes XDR, an example of what might be called a TCP/IP presentation protocol. Clark discusses passing information upward through layers [Clark 1985].

EXERCISES

11.1 Study the ISO layering model in more detail. How well does the model describe communication on a local area network like an Ethernet?

11.2 Build a case that TCP/IP is moving toward a five-level protocol architecture that includes a presentation layer. (Hint: various programs use the XDR protocol, Courier, and ASN.1.)

11.3 Do you think any single presentation protocol will eventually emerge that replaces all others? Why or why not?

11.4 Compare and contrast the tagged data format used by the ASN.1 presentation scheme with the untagged format used by XDR. Characterize situations in which one is better than the other.

11.5 Find out how UNIX systems uses the *mbuf* structure to make layered protocol software efficient.

11.6 Read about the System V UNIX *streams* mechanism. How does it help make protocol implementation easier? What is its chief disadvantage?