INTERNETWORKING WITH TCP/IP

Module 1 -Syllabus

Introduction & Overview – Motivation for internetworking, TCP/IP internet, Internet Services. Internetworking Concept and Architectural Model – Introduction, Application-Level Interconnection, Network-Level Interconnection, Properties Of The Internet, Internet Architecture, Interconnection Of Multiple Networks With IP Routers.

Protocol Layering- Introduction, The Need For Multiple Protocols, The Conceptual Layers Of Protocol Software, Functionality Of The Layers, ISO 7-Layer Reference Model, The TCP/IP 5-Layer Reference Model, Mapping Internet Addresses To Physical Addresses (ARP)

Internetworking

For over two decades, a new technology has evolved that makes it possible to interconnect many disparate physical networks and make them function as a coordinated unit. The technology, called internetworking, accommodates multiple, diverse underlying hardware technologies by providing a way to interconnect heterogeneous networks and a set of communication conventions that makes them interoperate. The internet technology hides the details of network hardware, and permits computers to communicate independent of their physical network connections.

The internet technology described in this book is an example of open system interconnection. It is called open because, unlike proprietary communication systems avail- able from one specific vendor, the specifications are publicly available. Thus, anyone can build the software needed to communicate across an internet. More important, the entire technology has been designed to foster communication among machines with diverse hardware architectures, to use almost any packet switched network hardware, to accommodate a wide variety of applications, and to accommodate multiple computer operating systems.

The TCP/IP Internet

U.S. government agencies realized the importance and potential of internet technology many years ago, and have funded research that has made possible a global Internet. This book discusses principles and

ideas underlying the internet technology that has resulted from research funded by the Advanced Research Projects Agency (ARPA). The ARPA technology includes a set of network standards that specify the details of how computers communicate, as well as a set of conventions for interconnecting networks and routing traffic. Officially named the TCP/IP Internet Protocol Suite and commonly referred to as TCP/IP (after the names of its two main standards), it can be used to communicate across any set of interconnected networks. For example, some corporations use TCP/IP to interconnect all networks within their corporation, even though the corporation has no connection to outside networks. Other groups use TCP/IP for communication among geographically distant sites.

It forms the base technology for the global Internet that connects over 170 million individuals in homes, schools, corporations, and government labs in virtually all populated countries. In the US, The National Science Foundation (NSF), the Department of Energy (DOE), the Department of Defense (DOD), the Health and Human Services Agency (HHS), and the National Aeronautics and Space Administration (NASA) have all participated in funding the Internet, and use TCPILP to connect many of their research sites. Known as the ARPANSF Internet, the TCP/IP Internet, the global Internet, or just the Internee, the resulting communication system allows subscribers to share information with anyone around the world as easily as they share it with someone in the next room. An outstanding success, the Internet demonstrates the viability of the TCP/IP technology and shows how it can accommodate a wide variety of underlying network technologies.

Internet Services

Protocols like TCP and IP provide the syntactic and semantic rules for communication. They contain the details of message formats, describe how a computer responds when a message arrives, and specify how a computer handles errors or other abnormal conditions. Most important, they allow us to discuss computer communication independent of any particular vendor's network hardware. A communication protocol allows one to specify or understand data communication without depending on detailed knowledge of a particular vendor's network hardware

Hiding the low-level details of communication helps improve productivity in several ways. First, because programmers deal with higher-level protocol abstractions, they do not need to learn or remember as many details about a given hardware configuration. Thus, they can create new programs quickly. Second, because programs built using higher-level abstractions are not restricted to a particular computer architecture or a particular network hardware, they do not need to be changed when computers or

networks are replaced or reconfigured. Third, because application programs built using higher-level protocols are independent of the underlying hardware, they can provide direct communication between an arbitrary pair of computers. Programmers do not need to build a special version of application software for each type of computer or each type of network. Instead, software built to use protocols is general-purpose; the same code can be compiled and run on an arbitrary computer.

Internetworking Concept And Architectural Model

The primary goal is a system that hides the details of underlying network hardware while providing universal communication services. The primary result is a high-level abstraction that provides the framework for all design decisions.

Application-Level Interconnection

Designers have taken two different approaches to hiding network details, using application programs to handle heterogeneity or hiding details in the operating system. Early heterogeneous network interconnections provided uniformity through application level programs called application gateways. In such systems, an application-level program, executing on each computer in the network, understands the details of the network connections for that computer, and interoperates across those connections with application programs on other computers For example, some electronic mail systems consist of mail programs that are each configured to forward a memo to a mail program on the next computer. The path from source to destination may involve many different networks, but that does not matter as long as the mail systems on all the machines cooperate by forwarding each message.

Using application programs to hide network details may seem natural at first, but such an approach results in limited, cumbersome communication. Adding new functionality to the system means building a new application program for each computer. Adding new network hardware means modifying existing programs (or creating new programs) for each possible application. On a given computer, each application program must understand the network connections for the computer, resulting in duplication of code.

Network-Level Interconnection

The alternative to providing interconnection with application-level programs is a system based on network-level interconnection. A network-level interconnection provides a mechanism that delivers small packets of data from their original source to their ultimate destination without using intermediate

application programs. Switching small units of data instead of files or large messages has several advantages. First, the scheme maps directly onto the underlying network hardware, making it extremely efficient. Second, network-level interconnection separates data communication activities from application programs, permitting intermediate computers to handle network traffic without understanding the applications that are sending or receiving it. Third, using network connections keeps the entire system flexible, making it possible to build general purpose communication facilities. Fourth, the scheme allows network managers to add new network technologies by modifying or adding a single piece of new network level software, while application programs remain unchanged. The key to designing universal network-level interconnection can be found in an abstract communication system concept known as internetworking. The internetwork, or internet, concept is an extremely powerful one. It detaches the notions of communication from the details of network technologies and hides low-level details from the user. More important, it drives all software design decisions and explains how to handle physical addresses and routes.

We begin with two fundamental observations about the design of communication systems:

(i)No single network hardware technology can satisfy all constraints.

(ii)Users desire universal interconnection.

Properties Of The Internet

The notion of universal service is important, but it alone does not capture all the ideas we have in mind for a unified internet because there can be many implementations of universal services. In our design, we want to hide the underlying internet architecture from the user. That is, we do not want to require users or application programs to understand the details of hardware interconnections to use the internet. We also do not want to mandate a network interconnection topology. In particular, adding a new network to the internet should not mean connecting to a centralized switching point, nor should it mean adding direct physical connections between the new network and all existing networks. We want to be able to send data across intermediate networks even though they are not directly connected to the source or destination computers. We want all computers in the internet to share a universal set of machine identifiers (which can be thought of as names or addresses).

Our notion of a unified internet also includes the idea of network independence in the user interface. That is, we want the set of operations used to establish cornrnunication or to transfer data to remain independent of the underlying network technologies and the destination computer. Certainly, a user

should not have to understand the network interconnection topology when creating or using application programs that communicate.

Internet Architecture

Physically, two networks can only be connected by a computer that attaches to both of them. A physical attachment does not provide the interconnection we have in mind, however, because such a connection does not guarantee that the computer will cooperate with other machines that wish to communicate. To have a viable internet, we need special computers that are willing to transfer packets from one network to another. Computers that interconnect two networks and pass packets from one to the other are called internet gateways or internet routers

Consider an example consisting of two physical networks shown in Figure 3.2 In the figure, router R connects to both network I and network 2. For R to act as a router, it must capture packets on network 1 that are bound for machines on network 2 and transfer them. Similarly, R must capture packets on network 2 that are destined for machines on network I and transfer them.

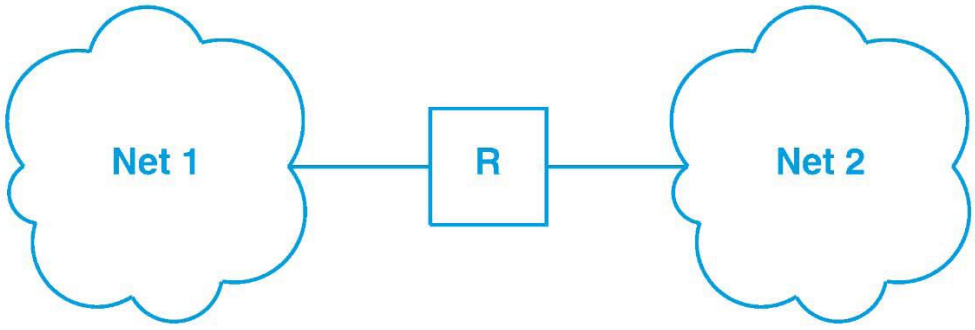

Figure 3.2 Two physical networks interconnected by an IP router, *R*.

In the figure, clouds are used to denote physical networks because the exact hardware is unimportant. Each network can be a LAN or a WAN, and each may have many computers attached or a few computers attached

Interconnection through IP Routers

In an actual internet that includes many networks and routers, each router needs to know about the topology of the internet beyond the networks to which it connects. For example, following Figure shows three networks interconnected by two routers.

In this example, router R1 must transfer from network I to network 2 all packets destined for computers on either network 2 or network 3. For a large internet composed of many networks, the router's task of making decisions about where to send packets becomes more complex.

The idea of a router seems simple, but it is important because it provides a way to interconnect networks, not just computers. In fact, we have already discovered the principle of interconnection used throughout an internet:

In a TCPBP internet, special computers called IP routers or IP gateways provide interconnections among physical networks.

You might suspect that routers, which must each know how to forward packets toward their destination, are large machines with enough primary or secondary memory to hold information about every computer in the internet to which they attach. In fact, routers used with TCP/IP internets are usually small computers. They often have little disk storage and modest main memories. The trick to building a small internet router lies in the following concept:

Routers use the destination network, not the destination computer, when forwarding a packet.

If packet forwarding is based on networks, the amount of information that a router needs to keep is proportional to the number of networks in the internet, not the number of computers.

Because routers play a key role in internet communication, we will return to them in later chapters and discuss the details of how they operate and how they learn about routes. For now, we will assume that it is possible and practical to have correct routes for all networks in each router in the internet. We will also assume that only routers provide connections between physical networks in an internet.

Protocol Layering

The Need For Multiple Protocols

Protocols allow one to specify or understand communication without knowing the details of a particular vendor's network hardware. 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 inverted arrivals. 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.

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.

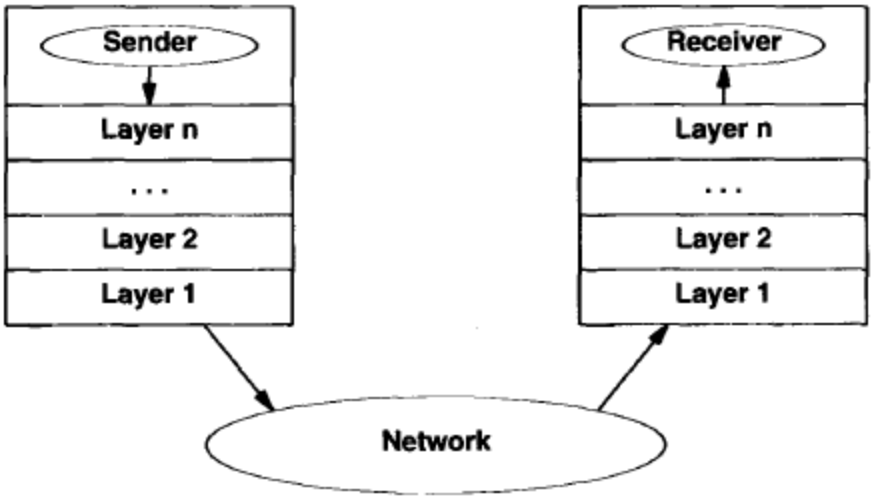

Figure 11.1 The conceptual organization of protocol software in layers.

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, forwarding 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 concept. 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 the routers because only those layers are needed to receive, route, and 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.

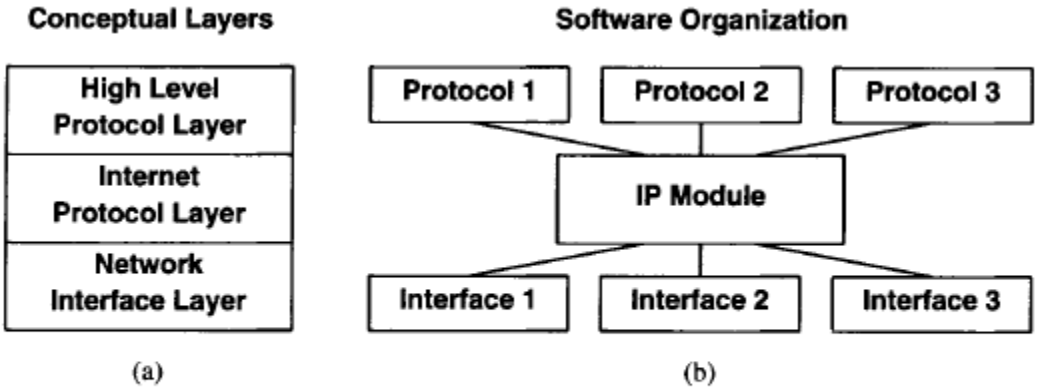

Figure 11.2 A comparison of (a) conceptual protocol layering and (b) a realistic view of software organization showing multiple network interfaces below IP and multiple protocols above it.

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 routers, the datagram passes up to the IP layer which sends 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.

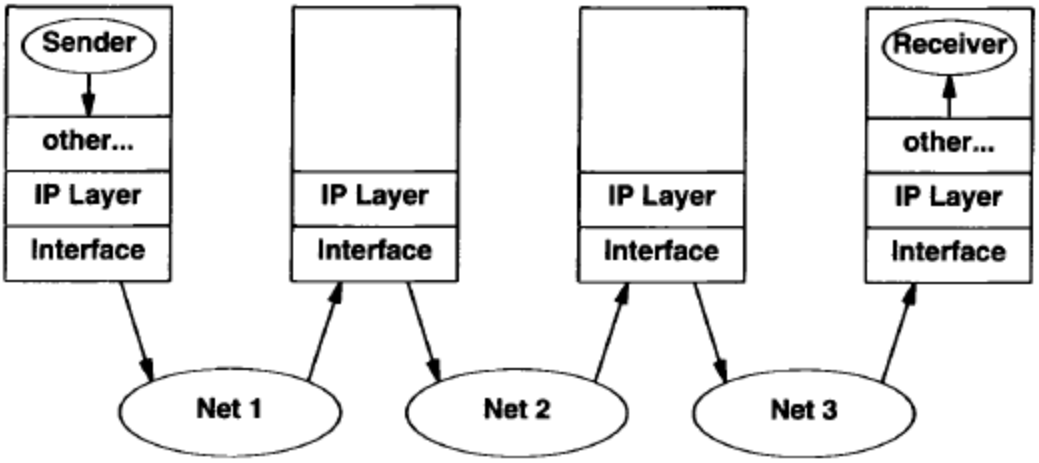

Figure 11.3 The path of a message traversing the Internet from the sender through two intermediate routers to the receiver. Intermediate routers only send the datagram to the IP software layer.

Functionality Of The Layers

IS0 7-Layer Reference Model

Two ideas about protocol layering dominate the field. The first, based on early 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 IS0 model. The IS0 model contains 7 conceptual layers organized as Figure 11.4 shows.

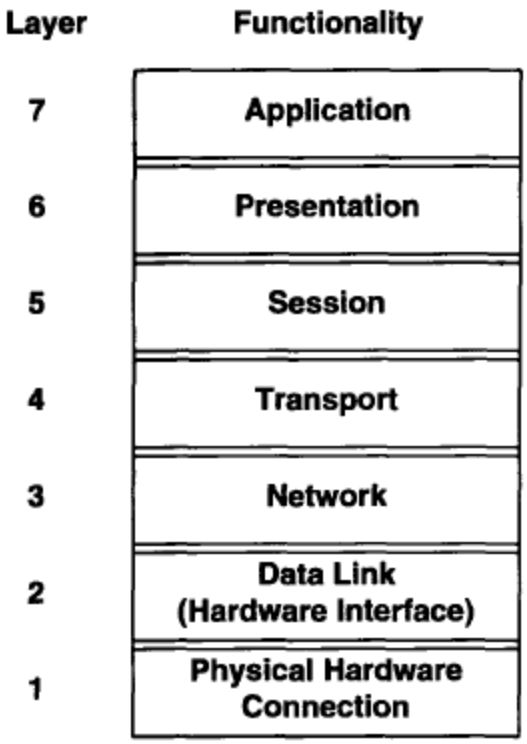

Figure 11.4 The ISO 7-layer reference model for protocol software.

Although it was designed to provide a conceptual model and not an implementation guide, the IS0 layering scheme has been the basis for several protocol implementations. Among the protocols commonly associated with the IS0 model, the suite of protocols known as X.25 is probably the best known and most widely used. X.25 was established as a recommendation of the International Telecommunications Union (ITU), formerly the CCIZT, an organization that recommends standards for international telephone services. X.25 has been adopted by public data networks, and became especially popular in Europe. Considering X.25 will help explain IS0 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, layer 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 layer 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 defined it). Because raw hardware delivers only a stream of bits, the layer 2 protocol must define the format of frames and spec@ how the two machines recognize frame boundaries. Because transmission errors can destroy data, the layer 2 protocol includes error detection (e.g., a frame checksum). Finally, because transmission is unreliable, the layer 2 protocol specifies an exchange of acknowledgements that allows the two machines to know when a frame has been transferred successfully.

One commonly used layer 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 HDLCLAPB. It is important to remember that successful transfer at layer 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.

Network Layer. The IS0 reference model specifies that the third layer contains functionality that completes the definition of the interaction between host and network. Called the network or communication subnet layer, this layer 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 layer 3 protocols to be larger than the size of frames that can be transferred at layer 2. The layer 3 software assembles a packet

in the form the network expects and uses layer 2 to transfer it (possibly in pieces) to the packet switch. Layer 3 must also respond to network congestion problems.

Transport Layer. Layer 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 layers of the IS0 model describe how protocol software can be organized to handle all the functionality needed by application programs. The IS0 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. IS0 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 IS0 standard known as Abstract Syntax Notation 1 (ASN.]), 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, IS0 layer 7 includes application programs that use the network. Examples include electronic mail or file transfer programs. In particular, the ITU has devised a protocol for electronic mail known as the X.400 standard. In fact, the ITU and IS0 worked jointly on message handling systems; the IS0 version is called MOTZS.

The TCP/IP 5-Layer Reference Model

The second major layering model did not arise from a standards committee, but came instead from research that led to the TCPIIP protocol suite. With a little work, the IS0 model can be stretched to describe the TCPAP layering scheme, but the underlying assumptions are different enough to warrant distinguishing the two.

Broadly speaking, TCPDP software is organized into five conceptual layers – four software 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.

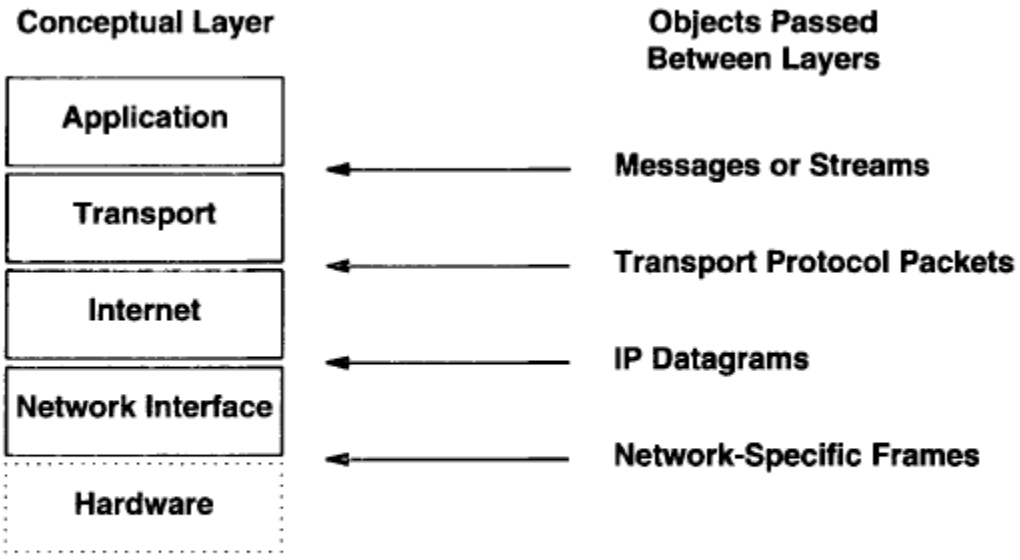

Figure 11.5 The 4 conceptual layers of TCP/IP software above the hardware layer, and the form of objects passed between layers. The layer labeled *network interface* is sometimes called the *data link* layer.

Application Layer. At the highest layer, users invoke application programs that access services available across a TCPDP internet. An application interacts with one of the transport layer 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 layer 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 an 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 deternune 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 and receives ICMP error and control messages as needed.

Network Interface Layer. The lowest layer TCPIIP 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).

Differences Between IS0 And Internet Layering

There are two subtle and important differences between the TCP/IP layering scheme and the ISOlX.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.

(i) 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 layers. 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 layer protocol includes timeout and retransmission algorithms that prevent data loss and provide automatic recovery after hardware fails and restarts.

(ii) 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 minimally in the network operation.

Mapping Internet Addresses To Physical Addresses (ARP)

Designers of TCPLP protocols found a creative solution to the address resolution problem for networks like the Ethernet that have broadcast capability. The solution allows new hosts or

routers to be added to the network without recompiling code, and does not require maintenance of a centralized database. To avoid maintaining a table of mappings, the designers chose to use a low-level protocol to bind addresses dynamically. Termed the Address Resolution Protocol (ARP), the protocol provides a mechanism that is both reasonably efficient and easy to maintain.

As Figure 5.1 shows, the idea behind dynamic resolution with ARP is simple: when host A wants to resolve IP address ZB, it broadcasts a special packet that asks the host with IP address le to respond with its physical address, PB. AU hosts, including B, receive the request, but only host B recognizes its IP address and sends a reply that contains its physical address. When A receives the reply, it uses the physical address to send the internet packet directly to B. We can summarize:

The Address Resolution Protocol, ARP, allows a host to find the physical address of a target host on the same physical network, given only the target's IP address.

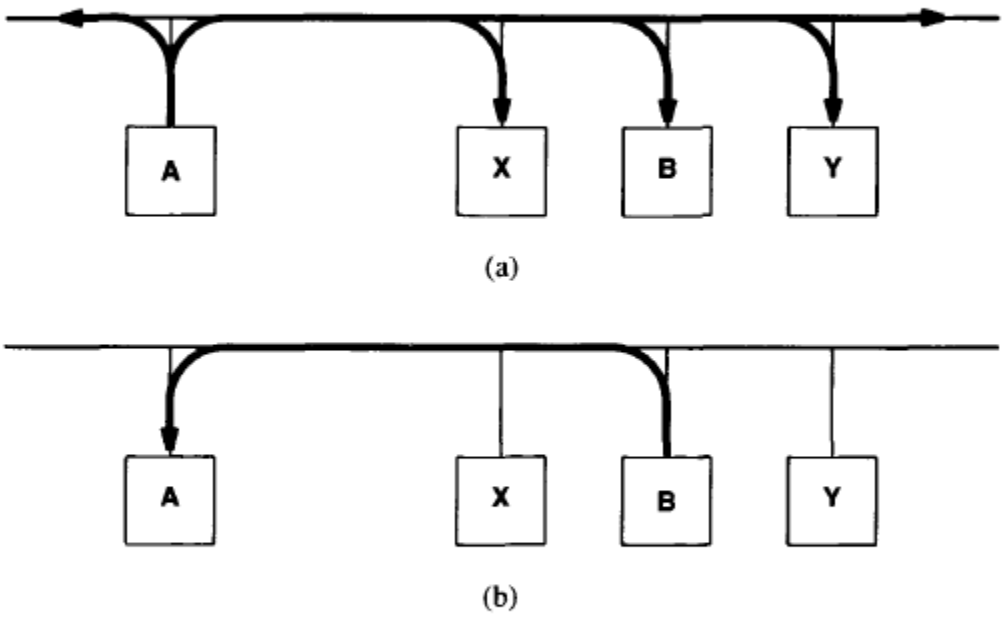

Figure 5.1 The ARP protocol. To determine P_B, B's physical address, from I_B, its IP address, (a) host A broadcasts an ARP request containing I_B to all machines on the net, and (b) host B responds with an ARP reply that contains the pair (I_B, P_B).

The Address Resolution Cache

It may seem silly that for A to send a packet to B it first sends a broadcast that reaches B. Or it may seem even sillier that A broadcasts the question, "how can I reach you?" instead of just broadcasting the packet it wants to deliver. But there is an important reason for the exchange. Broadcasting is far too expensive to be used every time one machine needs to transmit a packet to another because every machine on the network must receive and process the broadcast packet.

ARP Cache Timeout

To reduce communication costs, computers that use ARP maintain a cache of recently acquired IP-to-physical address bindings. That is, whenever a computer sends an ARP request and receives an ARP reply, it saves the IP address and corresponding hardware address information in its cache for successive lookups. When transmitting a packet, a computer always looks in its cache for a binding before sending an AFW request. If it finds the desired binding in its ARP cache, the computer need not broadcast on the network. Thus, when two computers on a network communicate, they begin with an ARP request and response, and then repeatedly transfer packets without using ARP for each one. Experience shows that because most network communication involves more than one packet transfer, even a small cache is worthwhile.

The ARP cache provides an example of soft state, a technique commonly used in network protocols. The name describes a situation in which information can become "stale" without warning. In the case of ARP, consider two computers, A and B, both connected to an Ethernet. Assume A has sent an ARP request, and B has replied. Further assume that after the exchange B crashes. Computer A will not receive any notification of the crash. Moreover, because it already has address binding information for B in its ARP cache, computer A will continue to send packets to B. The Ethernet hardware provides no indication that B is not on-line because Ethernet does not have guaranteed delivery. Thus, A has no way of knowing when information in its ARP cache has become incorrect.

To accommodate soft state, responsibility for correctness lies with the owner of the information. Typically, protocols that implement soft state use timers, with the state information being deleted when the timer expires. For example, whenever address binding information is placed in an ARP

cache, the protocol requires a timer to be set, with a typical timeout being 20 minutes. When the timer expires, the information must be removed. After removal there are two possibilities. If no further packets are sent to the destination, nothing occurs. If a packet must be sent to the destination and there is no binding present in the cache, the computer follows the normal procedure of broadcasting an ARP request and obtaining the binding. If the destination is still reachable, the binding will again be placed in the ARP cache. If not, the sender will discover that the destination is off-line.

The use of soft state in ARP has advantages and disadvantages. The chief advantage arises from autonomy. First, a computer can determine when information in its ARP cache should be revalidated independent of other computers. Second, a sender does not need successful communication with the receiver or a third party to determine that a binding has become invalid; if a target does not respond to an ARP request, the sender will declare the target to be down. Third, the scheme does not rely on network hardware to provide reliable transfer. The chief disadvantage of soft state arises from delay - if the timer interval is N seconds, a sender may not detect that a receiver has crashed until N seconds elapse.

ARP Protocol Format

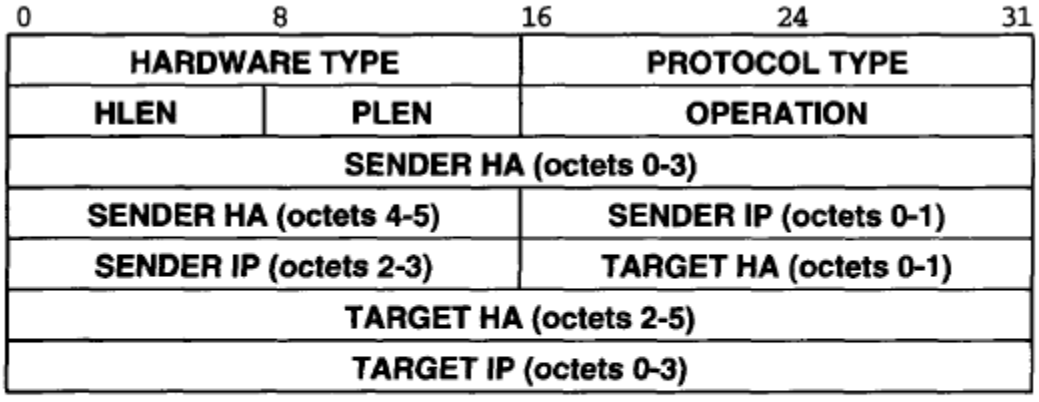

Figure 5.3 An example of the ARP/RARP message format when used for IP-to-Ethernet address resolution. The length of fields depends on the hardware and protocol address lengths, which are 6 octets for an Ethernet address and 4 octets for an IP address.

Field HARDWARE TYPE specifies a hardware interface type for which the sender seeks an answer; it contains the value 1 for Ethernet. Similarly, field PROTOCOL TYPE specifies the type of high-level protocol address the sender has supplied; it contains 0800,, for IP addresses. Field OPERATION specifies an ARP request (I), ARP response (2), RARP request (3), or RARP response (4). Fields HLEN and PLEN allow ARP to be used with arbitrary networks because they speclfy the length of the hardware address and the length of the high-level protocol address. The sender supplies its hardware address and IF' address, if known, in fields SENDER HA and SENDER IP.

When making a request, the sender also supplies the target hardware address (RARP) or target IP address (ARP), using fields TARGET HA or TARGET IP. Before the target machine responds, it fills in the missing addresses, swaps the target and sender pairs, and changes the operation to a reply. Thus, a reply carries the IP and hardware addresses of the original requester, as well as the IP and hardware addresses of the machine for which a binding was sought.